MICRO APPLICATION

LE LIVRE DU LOGO

AMSTRAD

PCW et CPC



UN LIVRE MICRO APPLICATION

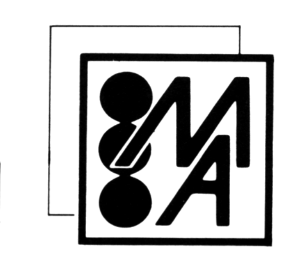
MICRO APPLICATION

LE LIVRE DU LOGO

AMSTRAD

PCW et CPC

UN LIVRE MICRO APPLICATION



Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

et

EDITION RADIO
9, Rue Jacob
75006 PARIS

ISBN : 2-86899-072-X

Traduction Française assurée par Pascal HAUSMANN

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

TABLE DES MATIERES

Nouveaux éléments de vocabulaire:
thing, list, listp, throw, TOPLEVEL

Programmes:
FILTRE (filtre de nombres primaires), TABLEAU, TABOUT, TABIN, TABPRINT (fonctions de tableau), ENTREE (programme de tableau), DEVINERBETES (Construction et extension de structures d'arbre), QUICKSORT (Programme de tri)

Nouveaux éléments de vocabulaire:
test, define, run, Structure de liste de programmes

Programmes:
LOGOPLAN (programme de table de calcul avec programmes auxiliaires), GENERATEURMASQUE (Générateur de programme)

PREFACE

Quand la ligne de salutation apparaît sur le moniteur, la programmation en LOGO peut immédiatement commencer. Vous pouvez commencer par provoquer directement, avec quelques mots, des actions simples. Ensuite vous définirez de la même façon un nouveau mot LOGO. Le programmeur peut ainsi accroître progressivement le vocabulaire du LOGO. Vous constaterez bientôt que le LOGO permet aussi de résoudre des problèmes complexes.

Les programmes LOGO se composent généralement de quelques lignes seulement. Les solutions des problèmes les plus complexes sont constituées à partir de petites unités, les procédures, qui ont chacune été formulées pour une tâche bien déterminée. La structure d'un programme LOGO reflète de ce fait directement la structure logique de la solution du problème.

En tant que langage de programmation, LOGO s'est particulièrement rendu célèbre par ce qu'on appelle Turtle graphics qui produit des effets graphiques à travers des instructions données à un crayon de dessin. Par la suite, cette méthode graphique a d'ailleurs également été utilisée dans d'autres langages de programmation. Le LOGO offre cependant des possibilités qui vont bien au-delà du graphisme de tortue (Turtle graphics); le but premier de cet ouvrage sera donc de guider l'utilisateur vers l'emploi de ces autres possibilités.

Les ordinateurs AMSTRAD sont fournis avec les deux langages de programmation LOGO et BASIC. Le PCW, en tant que système de traitement de texte est en outre doté également d'un programme de traitement de texte.

Parmi les acheteurs de micro-ordinateurs, la proportion de ceux qui utiliseront l'ordinateur essentiellement en tant qu'utilisateurs de systèmes de programmes tout prêts ne fera que s'accroître par rapport à ceux qui utilisent l'ordinateur pour la programmation. Le

LOGO, en tant que langage de programmation orienté de façon conséquente sur la solution des problèmes et en tant que langage interactif apparaît véritablement comme prédestiné pour l'utilisateur qui cherche surtout à pouvoir se faire une idée de ce qu'est la programmation mais qui, pour le reste, souhaite essentiellement utiliser des programmes d'application tout prêts. Les possibilités du LOGO sont cependant tellement diverses qu'il pénétrera ainsi toujours plus avant dans le domaine de la solution des problèmes avec un ordinateur.

Le développement de systèmes LOGO pour micro-ordinateurs n'a vraiment commencé qu'après 1980. Il n'existe pas encore une définition de ce langage qui fasse autorité. On peut certes le regretter mais cela permet par ailleurs à ce langage de continuer à évoluer rapidement. Il existe actuellement principalement deux familles de LOGO sur microordinateurs. Le DR LOGO pour les ordinateurs AMSTRAD vient de la firme de logiciels Digital Research (DR) et il ne se distingue naturellement des versions réalisées par le même fabricant pour d'autres ordinateurs que par la richesse de son vocabulaire et par différents éléments particuliers à cette machine.

Ces versions du LOGO sont apparentées à celles de la société LCSI qui ont reçu une large diffusion notamment sur les ordinateurs Apple et l'IBM PC, y compris les innombrables ordinateurs compatibles. Le présent ouvrage peut donc être en grande partie utilisé également comme une introduction au LOGO LCSI. Cette famille de LOGO se caractérise par des particularités telles que les listes de propriétés et la possibilité de sortie programmée des erreurs. Pour représenter ces caractéristiques, nous pouvons citer les mots LOGO PPROP et CATCH.

La seconde famille LOGO est généralement appelée MIT-LOGO. Cette famille est représentée par le LOGO pour le Commodore 64 et par la version exploitée par la firme IWT pour l'APPLE II. Cette famille LOGO ne se distingue pas, par ses instructions de base, de la précédente. Le vocabulaire est nettement plus restreint mais cette version a été réalisée de façon très efficace. Le présent ouvrage convient moins bien pour l'étude de cette version.

Ce livre a pour but de vous apprendre progressivement ce langage de programmation et ses possibilités d'application. Il est organisé en 19 leçons. Les douze premières leçons présentent les principaux éléments de vocabulaire par des exemples simples. Les sept dernières leçons expliquent des applications avancées par des exemples plus complexes. Les premières leçons ne nécessitent pas de connaissances préalables. Les lecteurs disposant déjà de certaines connaissances peuvent passer assez rapidement sur ces leçons.

Du fait de la facilité de la manipulation du graphisme sous LOGO, les applications graphiques constituent un élément important de l'initiation au LOGO. A cet égard, nous présenterons aussi bien des programmes élémentaires de Turtle-graphics que des exemples plus ambitieux tels que ceux s'attaquant au graphisme en trois dimensions.

Les applications de traitement de listes vont également des manipulations de mots simples aux applications avec des structures de données plus complexes. On traitera enfin des possibilités de gestion de fichier sous LOGO. Dans l'avant-dernier chapitre, un système complet de programmes de gestion de fichier sera développé.

Pour faciliter une vue d'ensemble de chaque chapitre, une liste des nouveaux éléments de vocabulaire présentés ainsi que des exemples de programmes expliqués vous est fournie au début de chaque leçon. Nous avons renoncé à une description systématique de tous les mots du LOGO parce que le manuel du fabricant suffit pour cela. A la fin de la plupart des leçons vous trouverez des exercices.

Nous souhaitons remercier ici le lectorat d'édition pour sa coopération dans la préparation de cet ouvrage.

Linden, juin 1986
Gerhard Sauer

Leçon 1: Le lancement de DR LOGO

Le LOGO vous permet à l'aide d'instructions simples d'effectuer des calculs, de réaliser des dessins mais également d'accomplir des tâches plus complexes telles que la gestion de fichiers. Pour que votre ordinateur dessine vraiment un cercle sur le moniteur lorsque vous entrez l'instruction correspondante, cette instruction doit être interprétée, c'est-à-dire qu'elle doit être correctement identifiée et convertie en instructions directes aux véritables processeurs.

Ce travail est effectué par l'interpréteur LOGO; sur votre ordinateur AMSTRAD, il s'appelle DR LOGO et il vient de la société de logiciels Digital Research. DR LOGO a cependant a son tour besoin de s'appuyer sur un système d'exploitation qui se chargera par exemple de traiter vos entrées au clavier ou de lire et d'écrire sur les disquettes. DR LOGO utilise donc le système d'exploitation CP/M (Control-Program for Microprocessors) de la même société.

Sur les ordinateurs CPC AMSTRAD vous vous trouvez après la mise sous tension de la machine dans un système d'exploitation intégré qui est directement lié au langage de programmation BASIC. Le PCW AMSTRAD se trouve par contre encore sans système d'exploitation après la mise sous tension.

La première étape du démarrage de LOGO consiste à charger le système d'exploitation CP/M. Sur les ordinateurs CPC, vous ne disposez sur les modèles les plus anciens, les 464 et 664, que de la version 2.2, alors que si vous avez un 6128 vous avez aussi la version 3.0 (CP/M Plus). La version de LOGO tournant sous CP/M 2.0 est malheureusement une version très réduite, sans doute en raison du manque de place mémoire. Nous décrirons ici systématiquement la version pour CP/M Plus; pour toutes les différences par rapport à cette version, nous vous invitons à consulter votre manuel d'utilisation.

Le CPC AMSTRAD

Pour le chargement de CP/M, placer la disquette système dans le lecteur de disquette et la lancer avec l'instruction:

SHIFT+@ CPM (Terminer avec la touche RETURN)

Après que vous ayez appuyé simultanément sur les touches SHIFT et @ un trait vertical apparaît sur le moniteur; il s'agit du caractère permettant d'appeler le lecteur de disquette. Une fois le système chargé, vous voyez apparaître en dernier sur le moniteur "l'interrogation" du système d'exploitation, sous la forme:

A>

La lettre désigne ici le lecteur de disquette standard, A. Le caractère ">" que vous voyez à la suite constitue une invitation à entrer des instructions.

Entrez maintenant:

SUBMIT LOGO3

Un programme de chargement portant le nom LOGO3.SUB sera alors mis en marche sur la disquette placée dans le lecteur de disquette. Ce programme préparatoire se charge également de permettre que le clavier puisse être utilisé correctement par la suite. Lorsque vous utilisez l'instruction SUBMIT, la disquette ne doit pas être protégée contre l'écriture, c'est-à-dire que l'orifice de protection contre l'écriture ne doit pas être ouvert.

Le nom LOGO3 distingue la version décrite ici de la version tournant sous CP/M 2.0. Comme nous n'utiliserons en principe que la version pour CP/M 3.0, il est conseillé de changer le nom du programme avec :

RENAME LOGO.SUB=LOGO3.SUB

L'instruction de lancement deviendra alors:

SUBMIT LOGO

Le chargement de DR LOGO est terminé lorsqu'apparaît "l'interrogation" LOGO; celle-ci est constituée par un point d'interrogation apparaissant au début de la ligne de l'écran, pour vous inviter à effectuer des entrées, suivi du curseur de l'écran qui indique où apparaîtra sur l'écran le prochain caractère entré.

Le PCW AMSTRAD

Le PCW a besoin qu'un système d'exploitation figure sur la disquette placée dans le lecteur de disquette standard immédiatement après la mise sous tension de la machine. Placez donc simplement dans le lecteur de disquette, immédiatement après la mise sous tension, la disquette CP/M Plus et le système d'exploitation sera chargé automatiquement. Si vous avez travaillé auparavant avec le système de traitement de texte et que vous vouliez lancer le système CP/M à partir de là, vous pouvez déclencher un nouveau démarrage de l'ordinateur avec la combinaison de touches

SHIFT+EXTRA+EXIT

D'après la description donnée dans le manuel d'utilisation, une fois CP/M chargé, vous devriez, comme sur le CPC, placer maintenant la disquette contenant le système LOGO dans le lecteur de disquette puis entrer l'instruction

SUBMIT LOGO

Avec la disquette système fournie à l'auteur du présent ouvrage, on obtient aussitôt le message d'erreur

SUBMIT?

En examinant le catalogue de la disquette, il s'avère que cette instruction CP/M manque sur la disquette. Si vous n'obtenez pas ce message d'erreur, c'est que vous disposez déjà d'une disquette plus récente, préparée de façon plus rationnelle. D'autres problèmes surgissent avec les jeux de caractères, aussi bien sur l'écran que sur l'imprimante connectée car le système PCW est réglé de façon standard sur le jeu de caractères français alors que le jeu de caractères américain est plus adapté pour le LOGO à cause des crochets.

Comme il est de toute façon vivement recommandé de ne travailler qu'avec une copie du système et non avec la disquette originale, le mieux est que vous commenciez par vous constituer une nouvelle disquette pour DR LOGO qui sera alors préparée correctement. Toute nouvelle disquette doit en premier lieu être formatée. Appelez pour cela le programme disckit sur la disquette système CP/M, en entrant

```
DISCKIT
```

et en suivant ensuite les instructions qui vous sont données par ce programme.

Placez ensuite la disquette système CP/M dans le lecteur de disquette et appelez le programme de copie PIP en entrant

```
PIP
```

Une étoile apparaît au début de la ligne comme message d'invitation. Copiez maintenant les programmes suivants l'un après l'autre sur la nouvelle disquette. Vous n'avez pas besoin pour cela de quitter le programme PIP.

```
*b:logo.com=logo.com
*b:keys.drl=keys.drl
*b:submit.com=submit.com
*b:setkeys.com=setkeys.com
*b:language.com=language.com
*b:type.com=type.com
*b:erase.com=erase.com
*b:setlst.com=setlst.com
```

Les deux premières parties (ce qu'on appelle des fichiers) figurent sur la disquette LOGO mais les autres se trouvent sur la disquette système CP/M. Lorsque vous entrez l'instruction de copie, la disquette source doit se trouver dans le lecteur de disquette; le système vous demande ensuite de placer dans le lecteur de disquette la nouvelle disquette (pour B).

Il est également recommandé de copier également le système d'exploitation CP/M sur la disquette préparée pour le LOGO (Fichier J14GCPM3.EMS) car vous n'aurez plus besoin alors que d'une seule disquette pour le lancement de DR LOGO.

Pour le lancement, un programme préparatoire LOGO.SUB doit être produit qui effectuera les réglages nécessaires du clavier, de l'écran et de l'imprimante et qui chargera ensuite le système LOGO proprement dit. Un tel fichier peut être réalisé avec l'éditeur de CP/M dont vous trouverez la description dans votre manuel mais dont le programme HELP sur la face 4 des disquettes système vous donne aussi une description rapide. Le plus simple est que vous suiviez très précisément les étapes décrites ci-dessous.

Entrez:

```
ed logo.sub
```

La disquette préparée doit se trouver dans le lecteur de disquette; vous devriez obtenir comme réponse le message NEW FILE et une "interrogation" constituée par une étoile. Tapez maintenant la lettre i, à la suite de quoi vous devriez voir sur l'écran

```
1:
```

Les lignes suivantes doivent maintenant être entrées l'une après l'autre et terminées chaque fois avec la touche RETURN:

```
1: language 0
2: setkeys keys.drl
3: setlst printer.drl
4: logo.com
```

Cette opération doit se terminer par la combinaison de touches ALT+Z. Après l'étoile qui apparaît maintenant, il faut appuyer sur la lettre e et ensuite sur la touche RETURN. La création du fichier peut maintenant être contrôlée avec l'instruction

```
type logo.sub
```

Si vous avez fait des erreurs, le mieux est de supprimer le fichier réalisé avec

```
erase logo.sub
```

et de le réécrire ensuite.

La seconde ligne du fichier appelle une routine système setkeys qui sert au bon réglage du clavier et qui utilise le fichier keys.drl, de la face 4 des disquettes système, que vous avez déjà copié. Comme les crochets jouent un rôle important sous LOGO, il est recommandé qu'ils apparaissent aussi en tant que tels sur l'écran et c'est pourquoi language 0 sélectionne le jeu de caractères américain.

> Les crochets peuvent alors être produits sur le clavier du PCW avec les touches ° et §.

Pour que les crochets soient également représentés correctement sur l'imprimante, celle-ci est basculée sur le jeu de caractères américain par la routine setlst. Il faut pour cela réaliser encore le fichier printer.drl qui n'est malheureusement pas fourni par le fabricant. Entrez donc, comme plus haut,

```
ed printer.drl
```

et procédez dans vos entrées comme pour le fichier LOGO.SUB. Une simple ligne suffit ici:

```
1: ^ 'ESC'R ^ 'NUL'
```

Cette ligne contient ce qu'on appelle une séquence Escape pour le réglage de l'imprimante.

Votre disquette devrait être maintenant tout à fait prête pour le lancement de DR LOGO sur le PCW. Après avoir entré

```
submit logo
```

les lignes du fichier LOGO.SUB devraient maintenant apparaître l'une après l'autre sur l'écran. Le message initial du LOGO apparaîtra enfin et un point d'interrogation apparaîtra en conclusion au début de la ligne pour signaler que le système LOGO est prêt à recevoir vos entrées.

Leçon 2: Le calcul avec LOGO

Nouveaux éléments de vocabulaire:
*, /, +, -, espace, make, nom, nom de procédure, sin, cos, arctan, round, int, quotient, remainder, édition sous LOGO

2.1 Premiers exercices de calcul

Le calcul a été la première fonction des ordinateurs. De nos jours, cependant, le calcul pur n'est plus qu'une des nombreuses fonctions utiles remplies par les ordinateurs. Les problèmes de calcul dans la vie professionnelle ou privée de tous les jours ne sont pratiquement plus jamais résolus sans l'aide d'une calculatrice de poche. Et d'ailleurs, les calculatrices de poche ne sont rien d'autre que des ordinateurs minuscules conçus pour cette utilisation spécifique.

Les microordinateurs tels que votre AMSTRAD ou tout autre ordinateur personnel ne se résument évidemment pas à des calculatrices de poche plus puissantes et plus pratiques. Leur intérêt vient au contraire de la gamme étendue de leurs utilisations possibles. Nous allons cependant examiner, pour commencer, ce que votre microordinateur peut faire dans ce domaine avec l'aide du LOGO. Pour de simples opérations de calcul, nous pouvons encore suivre le travail de l'ordinateur et en cas de doute nous pouvons toujours contrôler sur la calculatrice de poche.

Entrez donc une opération de calcul simple, par exemple:

```
3 * 5
```

Ce que vous venez de taper apparaît alors sur l'écran. L'exercice est ainsi formulé, le traitement par l'ordinateur pourra commencer dès que vous le lui aurez demandé en appuyant sur la touche RETURN. Sur la calculatrice de poche, c'est la touche '=' qui est utilisée à cet effet.

LOGO vous répond sur la ligne suivante:

```
15

(RESULTAT: 15)
```

Bien sûr, un problème tel que 3*5 est plus vite résolu de tête. Il n'en serait pas de même pour un problème tel que:

```
723+3*(27+28*(411+198)-2*78)
```

Peut-être aviez-vous toutefois trouvé tout de suite de tête le résultat 51492?

Dans un ouvrage de mathématiques, cet exercice se présenterait certainement sous la forme suivante:

```
723+3(27+28(411+198)-2*78)
```

En informatique cependant, il est absolument indispensable d'indiquer explicitement, avec *, toutes les opérations de multiplication ainsi que les autres opérations de calcul. Si vous oubliez seulement le premier signe de multiplication devant les parenthèses, LOGO répondra:

```
726

16923
```

L'ordinateur a d'abord exécuté l'opération 723+3. Comme aucun opérateur arithmétique ne suit, l'opération de calcul est considérée comme terminée et le résultat en est sorti sur l'écran. Le reste de la ligne constitue à nouveau une opération de calcul tout à fait correcte. Celle-ci sera résolue séparément puis son résultat apparaîtra sur l'écran.

S'il manque également le second signe de multiplication, le message suivant apparaît:

```
I don't know what to do with (
```

L'expression entre parenthèses ne peut pas être traitée sans le signe d'opération! Comme le reste de la ligne est marqué par les parenthèses extérieures comme constituant un exercice solidaire, le début, 27+28 ne peut être considéré comme une sous-opération valable, ce qui entraîne le message d'erreur.

L'oubli du signe de multiplication n'est qu'une des nombreuses erreurs et fautes de frappe qui se produisent encore souvent même lorsqu'on a déjà une grande expérience de l'emploi des ordinateurs. Lorsque vous faites une faute de frappe sur une calculatrice de poche, il vous faut en général recommencer à écrire l'opération tout entière si du moins vous avez remarqué l'erreur. Avec votre microordinateur, vous pouvez corriger l'écriture de l'opération tant que vous n'avez pas encore appuyé sur la touche RETURN.

Ce qui apparaît d'abord sur l'écran, c'est la ligne d'entrée, c'est-à-dire la ligne que vous devez entrer et qui doit comporter le problème de calcul. Cette ligne est mise à votre disposition d'abord comme une sorte de papier de brouillon. En effet, ce n'est qu'après que vous aurez appuyé sur la touche RETURN que LOGO prendra le contenu de cette ligne au sérieux. Mais ensuite, il n'y aura plus de pardon, toute erreur sera punie!

La plupart des opérations de calcul rentrent sur une seule ligne surtout si vous disposez d'un PCW avec son écran particulièrement large. Sur le CPC, il arrivera plus souvent qu'une opération dépasse la longueur d'une ligne car DR LOGO travaille ici en mode 40 colonnes. Lorsque votre entrée arrive au bord droit, l'écriture se poursuit automatiquement sur la ligne suivante de la fenêtre de texte. Le dépassement de la ligne est matérialisé à droite par un point d'exclamation. Les opérations de calcul peuvent cependant occuper encore beaucoup plus d'une ligne. Vous pouvez déterminer vous-même, par tâtonnement, la longueur maximum autorisée. A un moment ou à un autre, l'ordinateur refusera l'entrée et vous l'indiquera par un bip.

Le point d'exclamation est aussi utilisé dans le texte, surtout lorsqu'on reproduit des programmes, pour indiquer que la prochaine ligne d'impression appartient encore à la même ligne logique. Ces points d'exclamation doivent être transmis lors de l'entrée sur

l'ordinateur. Notez bien que la largeur de la ligne d'impression ne correspond pas à celle de la ligne de l'écran.

Une fois que LOGO a effectué l'opération de calcul, celle-ci est encore disponible dans l'ordinateur car elle a été transmise à une mémoire buffer, c'est-à-dire à une mémoire intermédiaire. Le contenu du buffer peut être réutilisé pour des entrées ultérieures. Il faut utiliser ici la combinaison de touches:

CONTROL+Y

Le signe plus ne doit pas être tapé, il sert simplement à indiquer ici que les deux touches doivent être actionnées simultanément. Sur l'écran apparaît à nouveau, en guise de réponse, le contenu de la ligne entrée en dernier. Cette ligne d'entrée apparaît dans la ligne actuellement traitée. Cela peut être utilisé pour une simple répétition de cette instruction mais aussi dans une ligne partiellement écrite pour ajouter le contenu du buffer. On peut continuer d'écrire la ligne même après CONTROL+Y. Vous pouvez aussi répéter CONTROL+Y plusieurs fois.

Sur le PCW AMSTRAD, la même fonction peut être également réalisée avec la touche COPY, COPY remplace donc CONTROL+Y. La touche que nous appelons ici CONTROL porte sur le PCW AMSTRAD l'intitulé ALT.

La combinaison CONTROL+Y déclenche une des nombreuses fonctions dites d'édition dont dispose le LOGO. La touche CONTROL est une touche de commutation, comme les touches Shift que vous connaissez certainement déjà puisqu'elles existent aussi sur la machine à écrire et qu'elles permettent de commuter entre majuscules et minuscules. La touche CONTROL modifie la signification de la touche utilisée. Pour le Y par exemple, aucun Y ne sera sorti à l'écran. Comme toutes les touches de commutation, la touche CONTROL doit être tenue enfoncée pendant que vous appuyez sur la touche Y qui déclenchera la fonction voulue.

2.2 Edition de la ligne d'entrée

Pour modifier une ligne d'entrée, le LOGO met à votre disposition ce qu'on appelle des fonctions d'édition. L'éditeur (c'est ainsi qu'on appelle la partie du système d'exploitation de l'ordinateur chargée de cela) mettra à votre disposition différents moyens, suivant la version du LOGO dont vous disposez. Les manipulations nécessaires ne seront pas non plus toujours identiques pour les mêmes fonctions. En cas de doute, il faut consulter le manuel du LOGO fourni par le fabricant. Il suffit parfois de tâtonner un peu pour trouver par soi-même les touches à utiliser. Dans le présent ouvrage, c'est l'éditeur du DR LOGO de l'AMSTRAD que nous décrirons; les fonctions d'édition ne sont décrites que de façon sommaire dans le manuel d'utilisation.

Mouvements du curseur

Le curseur, ou pointeur, indique où devra apparaître sur l'écran le prochain caractère tapé au clavier. Il peut être déplacé au moyen des touches de commande du curseur que l'on reconnaît aux symboles de flèches qu'elles portent. Ces touches se trouvent dans la petite zone de touches à côté du clavier des lettres.

Lors de la modification de lignes d'entrée, vous restez à l'intérieur d'une ligne (logique), de sorte que vous avez besoin ici essentiellement des touches pour déplacer le curseur vers la droite ou vers la gauche. Dans une ligne d'entrée, les déplacements verticaux ne sont possibles que si la ligne (logique) d'entrée s'étend sur plusieurs lignes matérielles de l'écran. Les déplacements du curseur ne sont possibles que dans la zone déjà remplie par des caractères.

Le LOGO met à votre disposition encore deux autres déplacements du curseur:

CONTROL+A	Le curseur est placé au début de la ligne d'entrée dans laquelle le curseur se trouve actuellement
CONTROL+E	Le curseur est placé à la fin de la ligne de l'écran

Notez que les déplacements du curseur se réfèrent à la structure de l'écran, pas aux lignes constituant une entité logique.

Sur le PCW, vous pouvez également utiliser la touche EOL/LINE pour amener le curseur à la fin ou au début de la ligne.

Insertion de caractères

Pour ajouter par exemple le signe de multiplication '*' qui manque dans la suite de caractères '3(', on placera le curseur sur la parenthèse et on tapera le signe de multiplication '*'. La parenthèse et tout ce qui la suit sera alors décalé automatiquement vers la droite. L'entrée de caractères au clavier a donc pour effet, sous LOGO, de faire insérer les caractères nouvellement entrés devant le caractère placé sous le curseur. Les caractères déjà existants ne seront pas effacés.

La touche 'Tab' (tabulateur) vous permet d'insérer de 1 à 4 caractères espace ce qui ne présente cependant pas un grand intérêt pour les lignes d'entrée. Les tabulations sont fixées en intervalles de 4 caractères (colonnes). La touche Tab place le curseur sur l'emplacement de la prochaine tabulation. Cela ne se fait cependant pas en sautant par dessus les caractères existants mais en insérant le nombre d'espaces nécessaire et en décalant le reste de la ligne s'il y en a un.

Suppression de caractères

LOGO offre à cet égard plusieurs possibilités:

La touche "DEL" (dans la zone principale, en haut, tout à droite)

Le caractère placé à gauche du curseur sera supprimé et simultanément le reste de la ligne d'entrée, à partir du curseur, sera ramené d'une position sur la gauche. En appuyant à plusieurs reprises sur la touche 'DEL', on déplace donc pas à pas vers la

curseur.

La touche "CLR" (à gauche de "DEL")
CONTROL+D avec la même fonction

Cela entraîne la suppression du caractère placé sous le curseur. Le reste de la ligne placé à la suite est ramené d'une position. L'emploi répété de cette fonction permet de supprimer la partie de la ligne d'entrée qui commence dans l'emplacement du curseur.

Si vous voulez supprimer une partie de la ligne, vous devez donc placer le curseur après le dernier caractère à supprimer si vous utilisez 'DEL' et devant le premier caractère à supprimer si vous utilisez CLR ou CONTROL+D.

Sur le PCW AMSTRAD, la touche CLR est également marquée "DEL". Pour les distinguer, la touche fonctionnant comme CONTROL+D comporte en plus une flèche vers la droite alors que la touche fonctionnant comme la touche "DEL" décrite ici comporte en plus une flèche vers la gauche.

CONTROL K

Cela permet de supprimer d'un seul coup tout le reste de la ligne, à partir de l'emplacement du curseur. CONTROL+A et CONTROL+K supprime une ligne toute entière.

CONTROL+K, au contraire de CONTROL+A et CONTROL+E, agit sur la totalité de la ligne logique ce qui n'est bien sûr pas tout à fait cohérent.

Correction de lignes d'entrée erronnées

Lorsqu'un message d'erreur apparaît à la suite de l'entrée d'une ligne, il faut d'abord que vous étudiez attentivement le message d'erreur. Amenez ensuite la ligne erronnée sur l'écran, avec la combinaison de touches CONTROL+Y (ou COPY sur le PCW) dont nous avons déjà parlé. Les fonctions d'édition dont nous avons parlé précédemment permettent de corriger les erreurs sans avoir à entrer à nouveau la totalité de la ligne concernée.

Les opérations de calcul sous LOGO

LOGO connaît bien sûr les quatre opérations de base:

Opération de calcul	Caractère	Exemple
Addition	+	2.1+3.8
Soustraction	-	7.2-5.5
Multiplication	*	3.8*7
Division	/	10/3

Les calculs de fractions ne peuvent être écrits sur les ordinateurs avec des traits de fraction car les nombres et les opérateurs arithmétiques doivent figurer sur la même ligne.

Les règles de priorité arithmétique s'appliquent: on calcule la multiplication et la division avant l'addition et la soustraction. Si on veut s'écarter de cette priorité, on doit l'indiquer en mettant entre parenthèses le calcul prioritaire. Les parenthèses permettent ainsi de transcrire aisément les calculs avec trait de fraction. Exemple:

```
(3.14*5*5+12)/(1.44-0.7)
```

Nous avons ici placé entre parenthèses le dénominateur d'une part et le diviseur d'autre part.

Notation avec infixe et avec préfixe

L'écriture habituelle:

```
3*5, 4-8, ...
```

d'une opération de base place l'opérateur arithmétique entre les deux opérandes. C'est ce qu'on appelle la notation avec infixe. Vous n'êtes certainement pas habitué à ce qu'on appelle la notation avec préfixe:

```
* 3 5 ou - 4 8
```

Dans ce système, on écrit en premier l'opérateur arithmétique suivi des opérandes. DR LOGO autorise systématiquement les deux modes d'écriture. Notez cependant que:

> dans la notation avec préfixe, l'opérateur et les opérandes doivent être chaque fois séparés entre eux par un espace.

Comme vous le constaterez plus tard lorsque vous pénètrerez plus avant dans l'étude des règles du LOGO, la notation avec préfixe correspond à la structure utilisée d'une manière générale pour les instructions LOGO: on indique tout d'abord ce qu'il s'agit de faire. Suivent ensuite des indications complémentaires nécessaires pour le problème donné. La notation infixée dans les opérations de calcul (ainsi que dans les opérations de comparaison) est une concession au mode d'écriture habituel en algèbre.

La notation préfixée présente l'avantage de pouvoir être également utilisée, sans risque de confusion, avec plus de deux opérandes. Essayez:

```
(+ 1 2 3 4)
```

Le résultat sera le nombre 10=1+2+3+4! Les parenthèses sont nécessaires pour que tout cela soit réuni dans une opération de calcul unique. On utilise toujours les parenthèses de cette manière, sous LOGO, lorsque le nombre d'entrées pour une opération

peut varier. On place entre parenthèses l'ensemble de l'opération si le nombre d'opérandes à suivre est plus grand ou plus petit que le nombre standard d'opérandes.

Pour les opérations telles que la soustraction, la division ou l'élévation à la puissance, il n'y a jamais que deux opérandes.

2.3 Stockage des nombres

Dans tous les domaines d'application, il y a des valeurs numériques dont on se sert souvent. En comptabilité on aura toujours besoin des taux de TVA, les physiciens auront besoin de constantes naturelles, les techniciens de mesures normalisées, les banquiers du cours du dollar, etc... Lorsque certaines valeurs numériques réapparaissent constamment dans des opérations de calcul, on aimerait ne pas avoir à la taper à nouveau chaque fois. Les calculatrices de bonne qualité permettent aussi de stocker certaines valeurs numériques pour qu'on puisse les rappeler lorsqu'on en a besoin.

En LOGO, on utilise pour cela l'instruction make:

```
make "TAUXTVA 14/100
```

Le sens de cette ligne est aisé à comprendre: le taux de TVA de 14% devra à l'avenir être disponible sous le nom de TAUXTVA. Les instructions de ce type sont appelées affectations de valeur. Les programmeurs BASIC connaissent pour cela le mot-clé LET.

Commençons par examiner la syntaxe, c'est-à-dire la structure grammaticale de cette ligne! La ligne commence par le mot-clé make (qu'on pourrait traduire ici par DEFINIR, FIXER) suivi du nom "TAUXTVA puis par la valeur numérique, ici sous la forme d'une division.

Notez que ces trois éléments doivent être séparés entre eux par des espaces! Le non-respect de cette séparation obligée ne sera plus toléré désormais. Essayez de voir ce qui se passe si vous oubliez un des deux espaces. Après:

```
make"TAUXTVA 14/100
```

le message "I don't know how to MAKE"TAUXTVA apparaît. Cela signifie: Je ne sais pas comment exécuter l'instruction MAKE"TAUXTVA ! Ce message vous indique que make et "TAUXTVA ont été considérés, en l'absence d'un espace de séparation, comme une expression solidaire.

Le taux de TVA peut bien sûr être également écrit directement sous la forme 0.14:

```
make "TAUXTVA 0.14
```

Vous savez certainement déjà, pour l'avoir constaté sur votre calculatrice, que notre 'virgule décimale' devient un point décimal en informatique, à cause bien sûr de l'influence des Etats-Unis dans ce domaine. Si vous négligez maintenant le second espace:

```
make "TAUXTVA0.14
```

le LOGO réagira avec le message d'erreur:

```
Not enough inputs to MAKE
```

En français: l'instruction make nécessite plus d'entrées.

Les groupes de caractères "TAUXTVA et 0.14 sont des entrées pour make. Les entrées nécessaires pour une instruction donnée sont toujours exécutées sous LOGO après l'appel de l'action demandée, selon le nombre et l'ordre prévus. Ces entrées doivent toujours être séparées entre elles par un espace. Si vous oubliez un espace, le nombre d'entrées ne correspond plus à celui attendu et vous obtenez le message d'erreur ...not enough inputs...

Les deux entrées "TAUXTVA et 0.14 sont de formes différentes; il n'y a rien de particulier à dire à propos de la valeur numérique 0.14. Mais pourquoi le taux de TVA est-il désigné "TAUXTVA et non simplement TAUXTVA?

Essayez:

```
make TAUXTVA 0.14
```

Le LOGO vous répond par un message d'erreur:

```
I don't know how to TAUXTVA
```

En français: je ne sais pas comment exécuter TAUXTVA. Le LOGO essaie visiblement d'interpréter TAUXTVA comme une instruction mais il ne peut pas l'exécuter. Les guillemets devant TAUXTVA sont nécessaires pour indiquer au LOGO que "TAUXTVA ne constitue pas une instruction.

make et "TAUXTVA sont des noms désignant des notions différentes. make désigne une action ou une instruction alors que "TAUXTVA doit remplacer une valeur numérique. Tous les noms qui ne désignent pas une action doivent être, sous LOGO, précédés d'un caractère introductif pour ne pas être confondus. Le nombre 0.14 sera correctement identifié par le LOGO, sans ajout particulier. Les nombres constituent également des mots pour le LOGO mais, du fait de leur signification spéciale, ils sont traités comme un type de mots particulier.

Majuscules et minuscules sous LOGO

Lors du lancement de DR LOGO, le clavier est réglé sur les ordinateurs AMSTRAD sur le mode normal, comme sur une machine à écrire, c'est-à-dire que les touches de lettres font apparaître tout d'abord des minuscules à moins que vous n'appuyiez simultanément sur la touche SHIFT pour produire des majuscules.

L'utilisation des majuscules et des minuscules est traitée différemment, sur les systèmes LOGO, en fonction des différents microordinateurs utilisés; en cas d'hésitation, il convient donc ici de faire quelques essais.

Avec le DR LOGO sur les ordinateurs AMSTRAD, il est possible d'écrire parallèlement en majuscules et en minuscules et le système LOGO remarquera effectivement la différence entre majuscules et minuscules. Les deux écritures

```
make "TAUXTva 0.14  et  make "TAUXTVA 0.13
```

doteront donc deux valeurs différentes d'un nom. Cela présente d'un côté l'avantage de vous offrir plus de possibilités différentes dans le choix des noms mais d'un autre côté l'utilisation simultanée de minuscules et de majuscules représente aussi une source d'erreur car vous devez vous souvenir non seulement du nom mais aussi de son écriture très précise en majuscules et minuscules afin d'utiliser chaque nom à bon escient.

Il est donc bien préférable de s'en tenir à une règle bien précise. Dans le présent ouvrage, nous écrirons le plus souvent en minuscules les noms qui, comme TAUXTVA, sont attribués à des valeurs. Les noms de programme seront par contre écrit en majuscules par souci de clarté. (Nous conserverons toutefois TAUXTVA pour la présente leçon).

Notez toutefois que:

> Les mots du vocabulaire propre du LOGO doivent absolument être écrits en minuscules. "MAKE" provoquera donc un message d'erreur.

Dans de rares circonstances, pour ce qu'on appelle les listes de propriété, DR LOGO utilise lui-même des majuscules mais nous vous l'indiquerons chaque fois de manière plus précise.

2.4 Calculs avec des nombres stockés

Revenons au calcul de la TVA. Comment devra se présenter une ligne d'entrée pour calculer la TVA d'un prix hors taxe de 50 F maintenant que le LOGO peut nous fournir le taux de TVA?

```
50 * :TAUXTVA
```

Quelle est la signification du double point devant :TAUXTVA et pourquoi n'avons-nous pas "TAUXTVA comme dans la ligne MAKE? Vous pouvez vérifier par vous-même que "TAUXTVA n'est pas correct car vous obtiendrez le message:

```
* doesn't like TAUXTVA as input
```

(L'opération * n'aime pas les entrées sous la forme "TAUXTVA.)

Vous connaissez donc maintenant trois façons pour indiquer un nom: sans caractère spécial introductif, avec guillemets introductifs et avec double point comme signe initial:

- Les noms sans caractère spécial désignent, sous LOGO, les instructions (ou actions, également appelées procédures).

- Les noms commençant par des guillemets désignent des noms auxquels on a affecté des propriétés particulières, des valeurs numériques par exemple. En mathématiques on parle de variables. Pour être plus concret on peut aussi dire qu'il s'agit de noms de tiroirs.

- Les noms qui commencent par un double point désignent la valeur représentée par un nom de tiroir. :TAUXTVA pourrait donc être ainsi traduit en clair: VALEUR DE TAUXTVA.

Autrement dit:

- "TAUXTVA indique le nom d'un tiroir,

- :TAUXTVA indique le contenu du tiroir considéré.

Votre microordinateur offre bien entendu de la place pour plus d'un tiroir. Si vous stockez par exemple également le prix hors taxe avec:

```
make "horstaxe 50
```

vous pourrez calculer, avec les lignes suivantes, la TVA et le prix net:

```
:horstaxe * :TAUXTVA
```

```
:horstaxe + :horstaxe * :TAUXTVA
```

Si vous avez déjà travaillé avec d'autres langages de programmation, il vous faudra vous habituer à l'emploi du double point sous LOGO pour désigner "la valeur de". Il vous arrivera certainement assez souvent au début de provoquer le message "I don't know how to...". Le fait que le LOGO ait absolument besoin qu'on lui indique nettement la différence entre le nom et le contenu d'un tiroir s'explique par certaines possibilités que possède le LOGO et qui font défaut dans la plupart des langages de programmation.

C'est ainsi par exemple que le contenu d'un tiroir peut être à son tour le nom d'un autre tiroir; les noms de tiroirs peuvent aussi être manipulés, c'est-à-dire décomposés ou recomposés. Les noms peuvent enfin recevoir encore d'autres propriétés que des valeurs. Nous n'étudierons cependant que plus tard ces possibilités sophistiquées du LOGO.

2.5 Les fonctions mathématiques

Les calculatrices ordinaires possèdent généralement une touche pour l'extraction de la racine, les calculatrices plus perfectionnées possèdent en plus des touches supplémentaires pour les fonctions trigonométriques et d'autres fonctions mathématiques. La version de DR LOGO pour les ordinateurs AMSTRAD est assez mal équipée dans ce domaine.

Fonction	Résultat	
round 3.2	3	Valeur arrondie du nombre 3.2
round 3.6	4	
int 4.3	4	Valeur entière du nombre 4.3
quotient 8 3	2	Valeur entière de la fraction 8/3
quotient 8 -3	-2	

remainder 11 4	3	Reste de la division de 11 par 4
remainder -10 3	-1	

Les fonctions round, int, quotient et remainder ont ceci en commun que leur résultat sera toujours un nombre entier. round arrondit la valeur en entrée au nombre entier supérieur ou inférieur. Pour les valeurs négatives en entrée, c'est toutefois la valeur absolue qui est ainsi arrondie. int fournit la partie entière d'une entrée.

quotient et remainder fournissent, lors de la division de nombres entiers, la partie entière du quotient et le reste de la division. Si les valeurs en entrée ne sont pas des nombres entiers, elles sont d'abord arrondies comme avec int. DR LOGO procède d'ailleurs toujours de cette manière: partout où une entrée entière est attendue, un nombre non-entier sera toujours rendu entier par suppression de la partie décimale.

En ce qui concerne l'écriture des constantes négatives, vous ne devez jamais oublier la différence entre le signe moins comme opérateur pour la soustraction et le caractère indiquant le signe d'un nombre.

DR LOGO interprète le signe moins comme opérateur lorsqu'il est suivi d'un espace. Pour que le signe moins soit interprété comme indication du signe, il ne doit donc pas être suivi d'un espace.

Attention: Toutes les versions LOGO ne fonctionnent pas exactement de cette façon!

La ligne suivante présente par exemple à cet égard une erreur fréquente:

```
quotient 8 - 3

Réponse: Not enough input to quotient
```

Le signe moins est interprété comme un opérateur, c'est-à-dire que le système calcule d'abord la valeur 5=8-3 qu'il utilise ensuite

comme entrée pour quotient. Mais comme quotient attend deux valeurs en entrée, cela entraîne un message d'erreur. Ce message est tout à fait caractéristique puisque du fait du calcul de la soustraction le nombre d'entrées est réduit de un, ce qui provoque l'erreur indiquée. La cause réelle de l'erreur est cependant simplement un espace de trop placé après le signe moins.

Si vous rencontrez à nouveau cette erreur par la suite, vous devrez vérifier si ce n'est pas un signe moins mal interprété qui vous a joué ce mauvais tour.

Nous avons utilisé les fonctions dans un premier temps pour produire le résultat immédiatement sur l'écran, comme marque de la réaction de l'ordinateur. Les résultats peuvent cependant également être engrangés.

```
make "nombre round 5.9
make "nombre round 5 / 3
make "nombre round :nombre
```

Une opération de calcul qui produira elle-même un nombre comme résultat peut aussi figurer à la place d'un nombre. Dans de telles opérations de calcul, vous pouvez également utiliser des fonctions.

Avec les fonctions avec une seule valeur en entrée, vous pouvez aussi placer celle-ci entre parenthèses et omettre alors l'espace de séparation:

```
round(5.9) round(:nombre/3)
```

Les parenthèses sont d'un usage courant pour les fonctions en mathématiques et elles sont aussi utilisées dans la plupart des langages de programmation.

Un second groupe de fonctions à notre disposition est constitué par les fonctions trigonométriques:

Fonction	Résultat	
sin 30	0.5	Sinus d'un angle de 30°

cos 30	0.866025	Cosinus d'un angle de 30°
arctan 0.5	26.565051	Tangente d'arc de 0.5 en degrés

Il n'y a rien ajouter au sujet de la signification de sin et cos. Les fonctions inverses sont représentées par la fonction de tangente d'arc à partir de laquelle peuvent être calculées les fonctions inverses des autres fonctions trigonométriques. Pour les fonctions trigonométriques, il faut noter que les angles doivent être indiqués en degrés ou qu'ils sont fournis en degrés par arctan.

Le grand nombre de décimales dans les résultats produits constitue une particularité des ordinateurs AMSTRAD. Cela crée l'illusion d'une précision qui fait totalement défaut. Le cosinus de 30 dégrés devrait en effet avoir par exemple la valeur 0.866025403784439 de sorte que seuls les sept premiers chiffres après la virgule se révèlent fiables.

Pour la conversion entre degrés et radians, on a besoin de la constante du cercle pi qui n'est pas intégrée dans ce système mais qu'on peut fort bien rendre disponible avec:

```
make "pi 3.14159265
```

Le conversion s'effectuera alors de la façon suivante:

```
make "degres :radians * 180 / :pi
make "radians :degres * :pi / 180
```

Exercices

1) Calculez l'expression

$$\frac{5x^2-3x}{2x + 1}$$

avec différentes valeurs pour x.

2) 1ère ligne: make "nombre pi make "nombreh 1
2ème ligne: make "nombreh :nombreh * :nombre :nombreh
Que se passera-t-il si vous entrez à nouveau la seconde ligne (avec CONTROL+Y)?

3) (make "a 5 sin :a) vous permet de réaliser une table mathématique de valeurs pour la fonction sinus en répétant cette formule et en remplaçant le nombre 5 par d'autres valeurs. Réalisez de la même manière une table pour la fonction de tangente (tan(x)=sin(x)/cos(x)).

4) 1ère ligne: make "w 1 make "r 2
2ème ligne: make "w (:w + :r / :w) / 2 :w
Recherchez ce qui se passera si vous répétez la seconde ligne.

Leçon 3: Les dés et les répétitions

Nouveaux éléments de vocabulaire:
random, rerandom, pr, type, copyon, copyoff, repeat, la notion de liste, interruption avec CTRL+G, ESC (STOP), CTRL+Z, co

3.1 Les nombres aléatoires (nombres tirés au hasard)

Il semble souvent contradictoire que le hasard puisse aussi jouer un rôle sur les ordinateurs. Pour vous en convaincre, entrez la ligne suivante:

```
random 6
```

Avec la touche de répétition (CONTROL+Y), vous pouvez entrer à nouveau cette ligne, sans la modifier. Bien que l'instruction soit toujours la même, le LOGO vous donnera effectivement des réponses chaque fois différentes. Le point commun entre ces divers résultats est que:

- Le résultat de random N est toujours un nombre entier.

- Sa valeur est toujours comprise entre 0 et N-1.

Le terme de nombre aléatoire indique que les résultats varieront parmi les valeurs possibles avec la même probabilité d'être tirée au sort pour chaque possibilité, comme on est en droit de l'attendre d'un véritable tirage au sort du style d'un lancer de dés non truqué. Toutefois, si on voulait imiter le lancer d'un dé, les résultats devraient varier de 1 à 6 et non de 0 à 5.

Cela peut être également obtenu, avec:

```
1 + random 6
```

Cette ligne d'instruction simule donc bien un lancer de dés honnête.

Chaque fois que vous relancez le LOGO, c'est la même suite de nombres aléatoires qui sera produite lorsque vous utiliserez random. Cet état initial peut également être restauré au moyen de

```
rerandom
```

Si notre ligne de simulation du lancer d'un dé commence par rerandom:

```
rerandom 1+random 6,
```

ce sera toujours le même nombre qui sortira. Certains ordinateurs permettent d'obtenir l'effet exactement inverse car ils veillent à ce que les suites de nombres aléatoires ne se répètent pas. On peut par exemple influencer de l'extérieur le début d'une suite de nombres aléatoires. Ce cas n'est pas prévu avec DR LOGO. On peut résoudre ce problème en prescrivant pour commencer un nombre variable de lancers de dé, c'est-à-dire un nombre modifiable d'appels de random.

3.2 Sortie des résultats

Jusqu'ici, le LOGO a traité toutes les opérations de calcul puis il en a indiqué le résultat. C'est très aimable de sa part car les résultats ne sont en effet obtenus qu'à l'intérieur de l'ordinateur. L'écran constitue pour l'ordinateur un périphérique de sortie exactement comme une imprimante ou un moyen de stockage externe tel que le lecteur de disquette. C'est pourquoi on doit, dans la plupart des langages de programmation, ainsi d'ailleurs que dans certaines versions du LOGO, toujours faire effectuer la sortie d'un résultat au moyen d'une instruction spécifique. Lorsque vous entrez une ligne d'instructions, DR LOGO sort automatiquement les résultats si cette ligne d'entrée peut être décomposée en opérations pouvant être interprétées correctement.

Les instructions qui sont données directement dans une ligne

d'entrée sont traitées d'une manière particulière par DR LOGO. LOGO essaie en effet de produire un maximum de résultats sensés, même si les règles de syntaxe qui s'appliquent normalement ne sont pas totalement respectées.

Pour sortir les résultats, on se sert normalement de l'instruction pr (abréviation de print). Voici quelques exemples:

```
pr 3*5

(pr 3*5 7*9)

pr 1+random 6
```

pr signifie "Sortir ... sur l'écran". Les données ou valeurs à sortir sont ensuite séparées entre elles par des espaces. Normalement, on attend un résultat lorsqu'on appelle PRINT. Si plusieurs exemples doivent être sortis, comme dans le second exemple, toutes les valeurs à sortir doivent être placées entre parenthèses, avec le mot-clé pr. Les parenthèses ont pour fonction sous LOGO de déclarer plusieurs choses solidaires.

Sortie sur imprimante

Si vous avez connecté une imprimante sur votre ordinateur, vous pouvez bien sûr également faire imprimer les résultats. Sous LOGO, c'est normalement très simple. Il suffit d'appeler

```
copyon
```

pour que toutes les sorties que vous ferez apparaître par la suite sur l'écran soient également copiées sur l'imprimante. copyon active l'imprimante comme second périphérique de sortie. La sortie sur imprimante peut être à nouveau désactivée avec

```
copyoff
```

3.3 Répétition d'instructions LOGO

On ne peut contrôler si 1+random 6 permet réellement de simuler les résultats d'un lancer de dé correct qu'à condition d'examiner de nombreux lancers de dé effectués par l'ordinateur. Bien sûr, la répétition de lignes d'entrée avec la combinaison de touches CONTROL+Y est déjà une méthode très simple que nous offre le LOGO. Toutefois un ordinateur peut très facilement réaliser lui-même la répétition d'une action, sans intervention manuelle de l'utilisateur. L'instruction repeat permet de demander à LOGO de répéter une action donnée. Pour dix répétitions par exemple:

```
repeat 10 [pr 1+random 6]
```

repeat a besoin de deux indications, d'abord combien de fois et ensuite ce qui doit être répété.

Le nombre de répétitions doit fort logiquement être un nombre entier. Le cas échéant, LOGO supprimera la partie décimale du nombre. LOGO autorise également le nombre 0. Dans ce cas, rien ne sera effectué dans la ligne de répétition. Le nombre de répétitions ne doit pas nécessairement être une constante. Vous pouvez utiliser ici la valeur d'une variable, par exemple

```
make "N 10
repeat :N [pr 1+random 6]
```

La constante peut aussi être remplacée par une opération de calcul si celle-ci produit un résultat qui puisse convenir. Le LOGO permet aussi de faire exécuter, dans un tel endroit, une action si cette action fournit un résultat utilisable, par exemple un nombre qui ne soit pas négatif. Essayez plusieurs fois l'exemple suivant:

```
repeat random 10 [pr 1+random 6]
```

random 10 tirera, ici, d'abord au hasard le nombre même des répétitions. Pour que le lien entre random et 10 apparaisse mieux à la lecture, vous pouvez placer ces deux éléments entre parenthèses:

```
(random 10).
```

Ce qui doit être répété correspond à ce qui constituait jusqu'ici une ligne d'entrée en soi. Vous pouvez faire répéter des actions et des suites de plusieurs actions. Dans notre exemple, l'action répétée est la sortie - pr - du résultat de l'opération de calcul 1+random 6. Notez que l'action pr doit absolument être indiquée ici. Ce n'est que pour les lignes d'entrée avec un résultat que le LOGO déclenche la sortie de lui-même.

L'action à répéter est enfermée entre crochets. Les crochets sont absolument indispensables et ils ne servent pas seulement à améliorer la présentation. Sans les crochets, vous obtiendriez le message [pr 1 + random 6] didn't output to repeat. Nous ne chercherons pas à savoir, pour le moment, pourquoi nous obtenons précisément ce message d'erreur.

Nous avons affaire ici à une seconde notion fondamentale du langage de programmation LOGO, la notion de LISTE.

La répétition avec repeat ne permet pas seulement de répéter une action mais aussi toute une série d'actions. Ces actions doivent être énumérées, séparées entre elles, comme nous y sommes déjà habitués, par des espaces. On les réunit en les enfermant entre crochets. Nous découvrirons plus tard encore toute une série d'autres possibilités de formation de listes.

Nous allons prendre l'exemple d'une amélioration de la répétition du lancer de dé pour illustrer une répétition de plusieurs actions.

La sortie des nombres tirés au sort s'effectuait jusqu'ici, chaque fois, dans une nouvelle ligne. L'instruction pr a toujours pour conséquence de produire automatiquement un passage à la ligne suivante, après la sortie. Si une sortie avec pr doit sortir plusieurs résultats, le LOGO produira automatiquement des espaces de séparation. Dans de nombreux cas, cela garantit ainsi une sortie lisible sans difficulté, sans qu'on ait à se préoccuper du format de la sortie. Les deux fonctions de formatage automatique, aussi bien le passage automatique à la ligne suivante que les espaces de séparation, peuvent cependant être interdites. A cet effet pr doit être remplacé par type.

```
type 3*5 type 7*9
```

```
Réponse: 1563
```

La réponse ne correspond bien sûr pas au nombre 1563; c'est plutôt que les deux résultats, 15 et 63 ont été sortis à la suite l'un de l'autre, sans séparation. Si les deux résultats sont sortis avec une seule instruction type, il y aura cependant une séparation par un espace.

```
(type 3*5 7*9)
```

```
Réponse: 15 63
```

Utilisons type, au lieu de pr, pour le lancer de dé répété:

```
repeat 10 [type 1+random 6]
```

Les dix nombres tirés au sort sont bien sortis sur une ligne, comme nous le voulions, mais sans caractères de séparation. Si nous renonçons, en employant type, au formatage proposé automatiquement par le LOGO, les signes de séparation doivent être prévus de manière explicite. On peut par exemple utiliser une virgule comme signe de séparation, à condition de l'indiquer expressément:

```
repeat 10 [type 1+random 6 type ",]
```

Pour sortir une virgule, il ne suffit pas d'écrire une virgule car alors le LOGO chercherait, en vain, une procédure portant ce nom. C'est pourquoi la virgule doit être précédée de guillemets.

Vous pouvez naturellement utiliser de même d'autres caractères spéciaux au lieu de la virgule. Pour la séparation avec des espaces, des mesures particulières sont nécessaires. Il en est toujours ainsi lorsqu'un caractère qui revêt une signification particulière dans la syntaxe du langage de programmation doit être traité comme un caractère ordinaire. La fonction 'grammaticale' de séparation du caractère espace est déconnectée lorsque vous le faites précéder d'un trait transversal (backslash):

```
repeat 10 [type 1+random 6 type "\ ]
```

Le symbole backslash (trait de fraction renversé) figure sur le clavier de votre CPC mais il peut aussi être produit par la combinaison de touches CONTROL+Q. Sur le PCW, vous ne disposez pas de cette touche, de sorte que vous devez utiliser la combinaison ALT+Q.

Le signe backslash lui-même n'est jamais sorti car son rôle consiste simplement à indiquer que le caractère suivant doit être considéré comme un caractère ordinaire. Les autres significations possibles, sous LOGO, du caractère placé après le backslash sont ainsi inhibées. Un espace ne sera par exemple plus considéré comme un signe de séparation.

Au lieu de répéter, par deux fois successives, l'instruction type, ces deux sorties peuvent également être réalisées avec une seule instruction type si celle-ci est placée entre parenthèses avec les entrées correspondantes:

```
repeat 10 [(type 1+random 6 "\ )]
```

En fait, un autre espace sera alors inséré dans ce cas. Au lieu de dix répétitions, nous allons nous autoriser une petite plaisanterie en demandant 10000 répétitions:

```
repeat 10000 [type 1+random 6 "# )]
```

Si vous en avez la patience, vous allez maintenant pouvoir réellement contrôler si le LOGO est à même, avec random, de simuler un véritable dé. Avec un bon dé, les nombres de sorties de chacun des chiffres 1 à 6 doivent avoir tendance à se rejoindre si on lance le dé suffisamment souvent.

A la longue, vous risquez tout de même de vous lasser d'assister à ces nombreux tirages au sort. Vous voyez, à cette occasion, qu'on a parfois besoin d'interrompre une action du LOGO pendant son déroulement.

L'interruption immédiate peut être provoquée de l'extérieur, c'est-

à-dire par l'utilisateur, en frappant une touche:

> ESC (STOP sur le PCW) arrête l'activité en cours du LOGO. Le LOGO vous répond par le message
>
> STOPPED!
>
> Au lieu de la touche ESC, vous pouvez également utiliser la combinaison CTRL+G.

Cette intervention interrompt totalement l'action entreprise, en l'occurrence les 10000 lancers de dé.

Il est également possible, lorsque cela présente un intérêt quelconque, de faire interrompre une action du LOGO de façon provisoire seulement. Il faut utiliser, pour cela aussi, une combinaison de touches spéciale:

> CTRL+Z interrompt le déroulement une action en cours. Cette action peut être poursuivie avec:
>
> co (abréviation de continuer).

Le LOGO indique l'interruption en sortant le message Pausing... Vous remarquerez, par la suite, que le LOGO donne en général encore d'autres indications, par exemple en quel endroit l'arrêt ou l'interruption s'est produit. C'est pourquoi des crochets apparaissent devant le point d'interrogation lors de l'exécution directe d'une ligne d'entrée.

Pendant la pause provoquée par l'interruption, d'autres actions peuvent être exécutées sans que cela remette en cause la possibilité d'une poursuite de l'action interrompue. Il est même possible de modifier l'instruction qui vient justement d'être interrompue! Il n'est vraiment mis fin à l'état d'interruption qu'avec co (Continue).

Exercices

1) On veut lancer simultanément deux dés. Le résultat sorti sera la somme des nombres tirés aux dés.

2) On sortira des dates de naissance de ce siècle, tirées au sort, avec le jour, le mois et l'année.

Leçon 4: La programmation

Nouveaux éléments de vocabulaire:
to, op

Programmes:
PERIMETRECERCLE, SURFACECERCLE

4.1 La programmation en tant qu'extension du langage

Travailler sous LOGO, cela signifie déclencher certaines actions bien précises du système LOGO. Une fois qu'est apparue la ligne de salutation, vous disposez d'un vocabulaire de base qui vous permet de donner des ordres. Pour un langage de programmation, il s'agit déjà là d'un vocabulaire très riche car LOGO vous offre déjà de nombreuses possibilités toutes prêtes. Mais ce qui est fascinant, lorsqu'on travaille avec des ordinateurs, c'est justement la possibilité de construire de nouvelles actions à partir de certaines actions de base simples. La diversité des actions imaginables est en principe illimitée.

Avant qu'une nouvelle action puisse être utilisée, le LOGO doit apprendre en quoi celle-ci doit consister. Programmer ne signifie rien d'autre qu'indiquer au système LOGO ce qu'il devra faire à l'avenir si une action, inconnue jusqu'ici, est appelée.

Sous LOGO, les actions inventées par l'utilisateur sont traitées exactement comme les possibilités déjà prévues dans le vocabulaire de base.

> Programmer en LOGO, cela signifie: étendre le vocabulaire du LOGO

Si la mémoire de votre ordinateur était illimitée, vous pourriez théoriquement, lors de chaque session de travail sous LOGO,

déclarer de nouvelles actions puis sauvegarder, à la fin de la session, le vocabulaire ainsi étendu sur disquette. Avant de reprendre le travail sous LOGO, vous pourriez donc, chaque fois, charger toutes les extensions du LOGO réalisées au cours des sessions précédentes; votre système LOGO ne ferait donc que s'accroître et la richesse du vocabulaire augmenterait sans cesse.

Bien entendu, vous ne disposez que d'un système limité. C'est d'ailleurs tout à fait suffisant car, en fait, pour une application donnée, on n'a besoin que d'une sélection d'actions requises pour cette application.

4.2 Le maniement des procédures

Les actions du LOGO sont aussi appelée procédures. Toutes les procédures sont traitées de la même manière par le système LOGO, aussi bien celles qui font déjà partie du vocabulaire de base du LOGO que celles qui ont été ajoutées par programmation. Pour comprendre comment des opérations peuvent être correctement déclenchées à l'aide de procédures, le plus simple est donc d'étudier les mots du vocabulaire LOGO que vous connaissez déjà, tels que pr, make, repeat, random etc...

On désigne d'abord l'action qui doit être exécutée. L'appel d'une procédure commence donc toujours par le nom de procédure correspondant. Ce nom doit être écrit sans caractère spécial introductif. Certaines actions n'ont besoin d'aucune autre indication.

Voici des exemples dont nous avons déjà parlé:

```
copyon, co
```

Dans la plupart des cas, les procédures nécessitent une ou plusieurs entrées. Voici encore quelques exemples dont vous connaissez déjà la signification:

```
pr "BONJOUR
make "n 2
```

```
repeat :n [pr 1+random 6]
```

Dans le premier exemple, les données à sortir avec pr doivent bien sûr être fournies, comme entrée, à l'action désignée sous le nom de pr. En l'occurrence, il s'agit de sortir le mot BONJOUR. Comme BONJOUR est un mot et non un nom de procédure, il faut placer des guillemets au début de ce mot, pour le distinguer des noms de procédure. Au lieu de "BONJOUR peut cependant également figurer un nombre, comme dans

```
pr 3.14
```

ou le nom d'une action qui fournisse un résultat à sortir, par exemple

```
pr random 6
```

Comme random nécessite à son tour une indication concernant la zone de nombres dans laquelle peuvent être puisés les nombres aléatoires, dans cet exemple, c'est l'appel complet random 6 qui remplacera le simple mot "BONJOUR du premier exemple. Le nombre 6 constitue ici une entrée pour random. Pour déterminer l'appartenance d'une entrée, il suffit de chercher, vers la gauche, le dernier nom de procédure indiqué immédiatement avant cette entrée.

Pour garantir une affectation correcte des entrées, lors de l'exécution d'une action, l'ordre des entrées doit être respecté strictement.
Le nombre des entrées est normalement fixé lui aussi et il doit être respecté. Quelques mots du vocabulaire LOGO, tels que pr ou type, font toutefois exception à cette règle.

pr nécessite normalement une entrée mais on peut toutefois également fournir plusieurs entrées, comme dans

```
(pr "pi=3.14159)
```

Lorsqu'on s'écarte du nombre standard d'entrées, l'ensemble de l'appel de procédure doit être inséré entre parenthèses.

Une entrée peut aussi être constituée par le contenu d'un tiroir, c'est-à-dire par la valeur d'une variable désignée par un mot:

```
pr :tauxtva
```

Le LOGO examinera alors s'il connaît le nom tauxtva. Si le programme ne trouve pas ce mot, le LOGO répondra:

```
tauxtva has no value (tauxtva n'a pas de valeur).
```

Si, par contre, le nom tauxtva avait bien été déclaré auparavant, le LOGO en sortira la valeur sur l'écran.

```
MAKE "tauxtva .14
PRINT :tauxtva
```

Notez bien la différence par rapport à:

```
PRINT "tauxtva
```

Dans ce cas, c'est le mot tauxtva qui sera sorti, tel quel, sur l'écran!

Examinons maintenant de plus près l'emploi de make dans un second exemple. make est une procédure à deux entrées: la première est un mot qui désigne un nom, la seconde est le contenu qui doit être placé dans le tiroir portant ce nom. La signification respective des entrées est clairement fixée par l'ordre dans lequel elles se présentent.

Essayez par exemple ce qui suit:

```
make "prenom "JEAN
pr :prenom
```

Le LOGO sort le mot JEAN, comme nous le voulions. Mais si vous échangez les deux entrées, dans l'instruction make, c'est exactement le contraire qui se passera: le mot prenom sera stocké sous le nom JEAN.

Un autre exemple, que nous connaissons déjà, nous permet de montrer une autre propriété importante des entrées. Dans la leçon précédente, nous avons placé le mot repeat au début de lignes d'entrée, pour en répéter l'exécution. repeat est une procédure à deux entrées:

```
repeat :n [pr random 6]
```

La seconde entrée est maintenant une LISTE, ce qui est bien marqué par les crochets entre lesquels elle est placée. A la place de cette liste pourrait aussi figurer le contenu d'un tiroir si ce contenu avait vraiment le type d'une liste.

L'exemple précédent pourrait également être remplacé par:

```
make "action [pr random 6]
repeat :n :action
```

Cet exemple constitue d'ailleurs en soi une propriété remarquable du LOGO! Lorsque vous appellerez repeat, il ne sera pas nécessaire de réécrire encore une fois la suite d'instructions à répéter. Le LOGO ira chercher les instructions comme la valeur d'un nom, c'est-à-dire dans un tiroir dont le contenu peut lui-même être manipulé. Une telle opération est tout à fait impossible dans la plupart des langages de programmation.

Avec repeat, on a affaire à deux entrées de types différents. La première entrée doit être un nombre ou un appel de procédure dont le résultat soit un nombre alors que la seconde entrée doit être une liste ou quelque chose qui débouche sur une liste. Si vous entrez un nombre à la place de la liste ou une liste à la place du nombre, par exemple:

```
repeat [10] [pr random 6]
```

le LOGO répondra par un message d'erreur:

```
repeat doesn't like [10] as input
```

4.3 Les procédures avec résultat

make déclenche une action dont on ne remarque pas l'effet immédiatement. make nécessite deux entrées mais aucun résultat n'est renvoyé sous la forme d'une sortie.

Il en va tout autrement lorsque vous entrez

```
random 6
```

à quoi le LOGO répondra par exemple:

```
1
```

random produit donc un résultat. C'est d'ailleurs le but recherché car cela n'aurait aucun intérêt que l'ordinateur s'amuse tout seul à tirer des nombres au hasard s'il ne les présentait pas aussi comme des résultats. Le plus souvent, on veut se servir de ces nombres pour exécuter encore d'autres actions, comme par exemple dans la ligne:

```
pr 1+random 6
```

On a ici ajouté encore 1 au résultat avant de sortir la somme obtenue. Inversement, dans la ligne:

```
pr make "n 2
```

aucun résultat n'est produit par make et la procédure de sortie pr ne trouvera rien à sortir. C'est pourquoi le LOGO répondra par le message d'erreur:

```
[make "n 2] didn't output to pr
(make n'a produit aucun résultat).
```

Les fonctions mathématiques fournissent également des exemples typiques d'actions avec sortie, par exemple:

```
sin 30
```

Il y a donc également des actions qui sortent comme résultat une valeur donnée qui pourra être ensuite traitée à son tour.

Notez à cet égard que:

-Une procédure peut avoir plusieurs entrées.
-Une procédure ne peut avoir, par contre, qu'une valeur De sortie ou pas du tout!

Les procédures qui fournissent exactement un résultat sont souvent appelées fonctions. On peut donc dire que le LOGO est un langage de programmation axé sur l'emploi de fonctions. Le fait qu'il ne puisse y avoir en résultat qu'une sortie et non plusieurs ne constitue toutefois une limite qu'au premier abord. Il est en effet possible de réunir plusieurs sorties dans une liste unique, de façon à ne former finalement qu'une seule valeur de sortie. Nous reviendrons en détail sur ce sujet lorsque nous parlerons du traitement des listes.

4.4 Ecriture de programmes simples

Après avoir décrit très complètement la maniement des procédures, grâce à quelques mots du vocabulaire de base du système LOGO, nous allons enfin nous attaquer maintenant aux premières extensions du langage.

Nous nous sommes jusqu'ici contentés de déclencher des actions du système LOGO et toujours des actions déjà définies. Lorsque nous voulons introduire de nouveaux mots de vocabulaire, le LOGO doit les apprendre. Par rapport au déclenchement d'actions déjà intégrées, cette opération constitue, au fond, un autre état du système. C'est pourquoi le LOGO doit être, à cet effet, placé dans un mode spécial d'apprentissage. Si vous avez déjà travaillé sur des calculatrices de poche programmables, vous connaissez déjà ce principe de la commutation vers un mode spécial.

Sur les ordinateurs, cette activité s'appelle l'édition de programmes. L'apprentissage ou l'édition peut concerner quelque chose de totalement nouveau mais cela peut aussi signifier qu'il

s'agit de modifier, notamment de corriger, quelque chose qui a déjà été appris. DR LOGO permet d'entrer une nouvelle procédure, ligne par ligne, dans la fenêtre de texte. Généralement, notamment pour la correction ou la modification de procédures déjà existantes, cela se fait dans une fenêtre particulière, qu'on appelle la fenêtre d'édition ou fenêtre Edit. Nous y reviendrons dans la leçon suivante.

Exemple 1: Le LOGO devra apprendre à lancer les dés. Vous connaissez, pour cela, déjà la ligne d'entrée:

```
1+random 6
```

L'instruction de lancer de dés est ici constituée à partir de mots déjà existants. Ce serait plus simple s'il suffisait d'entrer l'instruction LANCERDE pour que le LOGO vous fournisse une réponse appropriée. Pour apprendre au LOGO l'activité du lancer de dé, entrez:

```
to LANCERDE
```

Le sens du mot anglais 'to', qui sert aussi simplement à former l'infinitif (marcher se dit 'to walk'), peut être assez bien rendu par le mot français 'pour'. La ligne ci-dessus pourrait donc être traduite ainsi: nous allons maintenant t'expliquer ce qu'il faut faire 'pour lancer un dé'.

Certains d'entre vous seront peut-être gênés par ce mélange linguistique entre des mots de la 'langue maternelle' du LOGO, l'anglais et des mots français. Il existe d'ailleurs effectivement aussi des versions du LOGO en langue française. Toutefois ce mélange de mots français et anglais est devenu parfaitement habituel, en dehors bien sûr du monde anglo-saxon, dans les langages de programmation même si cela peut choquer un linguiste distingué.

Les mots-clé des langages de programmation d'aujourd'hui sont systématiquement empruntés à la langue anglaise, de même par exemple qu'on utilise un 'anglais sommaire' pour se comprendre dans le trafic aérien. Si on voulait faire une version de chaque langage

de programmation adaptée à chacune des langues naturelles de la planète, cela entraînerait plus de problèmes et surtout de coûts que d'avantages.

Pour le LOGO cependant, l'intérêt d'une version en français apparaît nettement plus que pour d'autres langages de programmation car le LOGO est considéré également comme un langage facile à apprendre par les enfants. D'un autre côté, le nombre de mots de la langue anglaise qu'il faut apprendre pour utiliser un langage de programmation est quand même très réduit. Le meilleur moyen d'apprendre ces mots consiste à les employer dans des applications pratiques. Ce n'est certainement pas la compréhension de quelques mots anglais élémentaires qui constitue la véritable difficulté de la programmation!

Mais revenons-en à l'apprentissage du mot-action LANCERDE!

Après que vous ayez entré la ligne to LANCERDE, en appuyant sur la touche RETURN, votre ordinateur vous indique qu'on passe à un autre état. Il modifie en effet 'l'interrogation', c'est-à-dire qu'il utilise un autre caractère pour vous indiquer qu'il est prêt à recevoir une entrée. Au lieu du point d'interrogation, c'est un > qui apparaît!

Entrez maintenant:

```
op 1+random 6
```

op signifie sortie. op indique que l'action 'LANCERDE' a un résultat, exactement comme, par exemple, random, cos, etc... op marque aussi, en même temps, la fin de l'action de LANCERDE. Il est donc inutile d'essayer d'écrire encore d'autres lignes, après la ligne avec op.

Nous avons donc fini d'expliquer LANCERDE. Il ne reste plus qu'à marquer la fin de la définition de cette action. Entrez pour cela:

```
end
```

DR LOGO remarquera alors que l'apprentissage de l'action

LANCERDE est terminé. Vous verrez en effet disparaître le signe > en début de ligne. Au lieu de cela, apparaît maintenant le message indiquant le succès de l'opération:

```
LANCERDE defined
```

(L'action de LANCERDE a été définie).

Le mot LANCERDE, c'est-à-dire la procédure portant le nom LANCERDE est donc maintenant disponible.

Veuillez noter toutefois que, lors de l'édition, le système LOGO ne contrôle pas si la déclaration de l'activité LANCERDE a bien été faite de façon correcte sur le plan de la syntaxe ni si cette déclaration correspond bien à une action quelconque. Les erreurs que pourrait donc recéler cette déclaration ne se révéleront donc que lors de l'exécution.

Contrôlez maintenant si LANCERDE vous permet réellement de lancer un dé. Il vous suffit pour cela d'entrer simplement ce mot. Vous constaterez alors que l'entrée de LANCERDE a effectivement le même effet que la ligne d'entrée 1+random 6!

D'autres exemples de programmes simples:

Calculs de cercles

De la même manière que pour la programmation du mot LANCERDE, nous pouvons introduire des programmes élémentaires de calcul du périmètre ou de la surface d'un cercle, sous la forme de nouveaux mots du vocabulaire LOGO :
PERIMETRECERCLE, SURFACECERCLE.

Après avoir entré 'to PERIMETRECERCLE', il vous faut indiquer, comme sortie de la procédure PERIMETRECERCLE, la formule de calcul du périmètre d'un cercle:

```
to PERIMETRECERCLE
op 2* :pi*:r
```

De même, pour la surface du cercle:

```
to SURFACECERCLE
op :pi*:r*:r
```

Les deux formules utilisent la constante du cercle que nous avions déjà définie sous le nom de pi avec:

```
make "pi 3.14159265
```

Si vous n'avez pas commis de faute de frappe, les deux nouveaux mots LOGO PERIMETRECERCLE et SURFACECERCLE devraient avoir le même effet que les lignes d'entrée correspondantes.

Il se peut cependant que le LOGO vous réponde par le message d'erreur:

```
I don't know how to r in PERIMETRECERCLE: op 2 * pi * r
```

Il n'y a pas d'action définie sous le nom de r; le LOGO ne sait donc pas comment exécuter 'r'. Le nom de procédure r est utilisé dans la procédure PERIMETRECERCLE, dans la ligne op 2 * pi * r. L'erreur apparue est facile à identifier: à la suite d'une faute de frappe, il manque le double point devant r, double point qui indiquerait au LOGO qu'il s'agit de la valeur de r et non d'une procédure r.

Comme vous le voyez, DR LOGO expose son point de vue de façon détaillée. Dans le message d'erreur, on vous indique où est apparu un problème et quel est ce problème. On vous montre également le texte de la ligne où est apparue l'erreur. Dans une telle situation, le système sait en quel endroit une difficulté a surgi. Vous pouvez en profiter en entrant:

```
ed
```

Le système LOGO réagira en passant dans un état spécial. Vous verrez ensuite apparaître le texte du programme PERIMETRECERCLE, le contenu antérieur de l'écran aura disparu et une ligne marquée

Edit

sera apparue tout en bas de l'écran. Dans ce cas, le curseur sera placé sur la lettre posant problème, à savoir r dans la seconde ligne.

Notez toutefois que ce que vous voyez maintenant vous indique précisément l'endroit où a été identifiée l'erreur qui a empêché la poursuite du travail normal du système LOGO. Il peut arriver aussi, de temps en temps, que la véritable erreur se soit produite plus tôt, notamment lorsqu'un programme a lui-même été appelé par un autre qui posait vraiment problème.

La prochaine leçon vous expliquera comment effectuer les corrections nécessaires et comment faire à nouveau accepter par le système LOGO le programme corrigé.

Leçon 5: Edition de programmes

Nouveaux éléments de vocabulaire:
ed, po, edall, pops, pons, pots, er, ern, erall, instructions d'édition, noms globaux et noms locaux

5.1 Correction de programmes

En principe, on peut corriger les programmes en les tapant une nouvelle fois, sans erreur. Ce n'est pas vraiment pratique et surtout cela comporte le risque de nouvelles erreurs. C'est pourquoi DR LOGO vous offre un éditeur pratique. L'éditeur est la partie du système LOGO qui est chargée de la mise en forme des textes de programmes.

Même lorsque vous écrivez de nouvelles procédures, il est recommandé d'écrire cette procédure d'emblée avec cet éditeur. Ce n'est que lorsque vous tapez une procédure toute prête et dont vous êtes sûr qu'elle ne nécessitera pas de corrections qu'il est en fait préférable d'écrire le texte du programme directement dans la fenêtre de texte.

L'éditeur est un programme de traitement de texte propre au LOGO. Il n'est cependant pas aussi pratique qu'un système de texte commercial prévu à cet effet. Il vous permet de modifier et de compléter les textes de programmes. Vous pouvez traiter isolément différentes procédures ou bien aussi simultanément tous les textes de programmes se trouvant dans la mémoire de travail de l'ordinateur. Vous pouvez même utiliser l'éditeur pour modifier les valeurs que vous avez affectées à des noms déterminés. En règle générale, nous vous conseillons toutefois de ne jamais éditer qu'une seule procédure à la fois.

On parvient dans le mode d'édition comme nous l'avons déjà indiqué, en entrant ed à la suite d'un message d'erreur. On commence le

traitement d'une procédure avec l'instruction ed:

```
ed "PERIMETRECERCLE
```

Il s'agit ici de traiter la procédure portant le nom 'PERIMETRECERCLE'. Notez que les guillemets devant PERIMETRECERCLE sont indispensables. Cela correspond à la règle qui s'applique aussi pour make. Par contre, lorsque vous entrez un texte de procédure dans la fenêtre de texte, avec to PERIMETRECERCLE, le nom de procédure s'écrit sans guillemets.

Après ed, la totalité de l'écran est mise à disposition pour l'édition. L'écran se transforme donc en fenêtre d'édition. La fenêtre de texte que vous pouviez voir auparavant est alors effacée et il en va par ailleurs de même pour la fenêtre graphique.

Si la procédure appelée existe déjà, le texte du programme apparaîtra sur l'écran. Si cette action est appelée pour la première fois, vous ne verrez apparaître que:

```
to PERIMETRECERCLE
end
```

ed doit être simplement suivi du nom de procédure avec guillemets introductifs. Le LOGO ajoutera alors de lui-même un end. Votre tâche consiste donc à entrer le texte du programme entre le nom de procédure et la fin de la procédure.

Lors d'une opération d'édition, vous pouvez écrire plusieurs procédures à la suite l'une de l'autre. Chacune doit se terminer par la ligne end. DR LOGO écrit d'avance un end et c'est à vous d'écrire les autres.

Comment peut-on ressortir du mode d'édition?

Vous devez utiliser pour cela la combinaison CONTROL+C.

-CONTROL+C met fin à l'édition.

Sur le CPC AMSTRAD, vous pouvez également utiliser à la place la

touche

COPY

(Le PCW dispose également d'une touche portant le même nom mais elle y a la fonction de CONTROL+Y).

Le LOGO réagira par le commentaire:

PERIMETRECERCLE defined
(PERIMETRECERCLE défini).

Si vous êtes totalement mécontent de l'opération d'édition, vous pouvez tout arrêter avec ESC (STOP). Vous annulez ainsi toute l'édition.

Mouvements du curseur

Lors de l'édition de lignes de programme, le curseur peut être déplacé dans les quatre directions, donc aussi vers le haut ou le bas. Le curseur peut, de plus, être amené au début de la ligne, avec CONTROL+A, ou en fin de ligne, avec CONTROL+E.

Si une ligne de programme est interrompue par le bord droit de l'écran, un point d'exclamation indique, à la fin de la ligne d'écran, que la suite se trouve sur la ligne suivante.

Une ligne de programme se termine en appuyant sur la touche RETURN. Les espaces qui apparaissent encore, à la suite, sur la ligne d'écran, ne font pas partie du texte du programme. La présentation des programmes LOGO peut être organisée grâce aux lignes. Cette présentation ne joue aucun rôle par rapport aux règles de syntaxe du LOGO mais elle permet de rendre le texte du programme plus facile à lire. En tout cas, il est recommandé d'éviter les longues lignes de programme lorsqu'elles ne sont pas indispensables.

Lorsque vous terminez une ligne d'écran par RETURN, la fin de la ligne est indiquée par l'emplacement actuel du curseur. S'il y a encore, sur l'écran, du texte de programme, à la suite du curseur,

le reste sera automatiquement déplacé vers la ligne suivante, ce que vous pouvez bien sûr également voir à l'écran. On peut, de cette façon, décomposer même une ligne de programme assez longue en plusieurs lignes.

L'éditeur LOGO autorise, de plus, des déplacements du curseur sur de grandes distances.

-CONTROL+R amène le curseur au début du texte du programme actuellement traité.

-CONTROL+X amène, au contraire, le curseur à la fin du texte traité.

-CONTROL+L place la ligne de texte actuellement appelée au milieu de l'écran. On peut ainsi rendre apparent le texte environnant une ligne.

-CONTROL+V fait passer le curseur à la page de texte suivante,

-CONTROL+U fait passer le curseur à la page de texte précédente.

Suppression

C'est CONTROL+D qui permet de supprimer le caractère placé sous le curseur alors que CONTROL+K supprime le reste de la ligne dans la fenêtre, en commençant à partir de l'emplacement du curseur.

La touche DEL supprime le caractère placé avant le curseur. Si le curseur se trouve au début d'une ligne, le symbole de séparation interne, qui sert à marquer, pour le LOGO, la fin d'une ligne, est supprimé. La ligne de programme au début de laquelle se trouve le curseur est donc ajoutée à la ligne précédente. On peut donc supprimer de cette manière la séparation entre deux lignes de programme qui avait été mise en place avec la touche RETURN.

Insertion de caractères et de lignes

Lors de l'édition de programmes, vous pouvez insérer des lignes entières. On entre pour cela un:

CONTROL+O (Ouvrir nouvelle ligne)

Le texte de programme suivant l'emplacement du curseur est ramené en arrière, de façon à faire de la place pour le début d'une nouvelle ligne.

L'insertion de caractères se fait comme pour les lignes d'entrée. Les caractères tapés sont écrits dans l'emplacement du curseur, le texte de programme existant est auparavant, chaque fois, décalé d'une position vers la fin du texte, de sorte qu'il n'est pas effacé. La combinaison de touches CONTROL+Y, dont nous avons déjà parlé pour les lignes d'entrée, permet d'insérer la section de texte se trouvant actuellement dans le buffer d'entrée, notamment le reste de la ligne au cas où il aurait été supprimé par erreur.

Dans le programme PERIMETRECERCLE de la leçon précédente, il nous a, par exemple, manqué une fois le double point devant le nom r (rayon). Pour corriger cela, nous plaçons le curseur sur le r et nous entrons le double point. CONTROL+C nous permet ensuite de faire entrer le programme ainsi corrigé dans le vocabulaire du LOGO, l'ancien texte du programme étant alors automatiquement supprimé.

5.2 Programmes avec entrées

A la leçon 2, nous avons vu un exemple de calcul de TVA où l'ordinateur exécutait le calcul immédiatement après l'entrée de la ligne d'instruction. Essayons maintenant de résoudre le même problème avec un programme:

```
to TVA
op :tauxtva * :prix
end
to prixnet
```

```
op (1 + :tauxtva)*:prix
end
```

Les deux procédures ont besoin de la valeur du taux de TVA qui devra donc avoir été stocké sous ce nom avec une ligne telle que:

```
make "tauxtva 0.14
```

Il est bien sûr très pratique de pouvoir indiquer ainsi, une fois pour toutes, au système LOGO la valeur de tauxtva. Cela vous évite en effet de devoir entrer vous-même ce nombre chaque fois.

Le nom tauxtva, qui désigne le taux de TVA, devrait donc pouvoir être accessible à toutes les actions qui se déroulent pendant cette session de travail sous LOGO. Des noms de ce type sont appelés noms globaux. Le terme global signifie ici qu'une fois fixés de tels noms font partie du système LOGO et que toutes les actions LOGO peuvent ensuite y accéder.

Peu importe qu'il s'agisse ou non, dans notre exemple, du véritable taux de la TVA. Un physicien aimerait certainement pouvoir disposer de constantes physiques sous des noms évocateurs. Il serait peut-être intéressant pour un médecin de conserver de cette manière le taux de sa rémunération à l'acte.

Pour que les procédures de calcul de la TVA puissent fournir une valeur, il faut bien sûr aussi que le prix soit connu. Dans la version actuelle de notre programme, il faudrait donc, avant d'appeler TVA ou PRIXNET, définir le prix:

```
make "prix 156.30
```

De même que le taux de TVA, le prix deviendrait ainsi un nom global. Toutes les actions ultérieures du LOGO pourront accéder, sous ce nom, au nombre 156.30, jusqu'à ce que sa valeur soit éventuellement modifiée par un nouvel appel de make avec le nom prix.

Il semble toutefois assez inutile et même assez incommode de traiter ainsi la valeur très particulière qu'est 156.30. Ce nombre

n'a en effet aucune signification générale qui justifie qu'on lui fasse une place fixe pour toutes les actions ultérieures du LOGO. Cette méthode est incommode parce qu'il nous faut, pour tout nouveau prix, entrer une ligne de la forme:

```
make "prix 520.10
```

Il serait très préférable de pouvoir employer une méthode voisine de celle utilisée pour les fonctions mathématiques, à savoir par exemple:

```
cos 30
```

Il suffit ici de modifier la valeur d'entrée 30 pour obtenir une nouvelle valeur pour la racine carrée. Ici aussi, le nombre 2 ne présente d'intérêt particulier qu'au moment de l'appel de la fonction. C'est pourquoi les programmes de calcul de la TVA seraient plus facile à manier si le prix actuel pouvait être entré sous la forme d'une entrée des procédures TVA ou PRIXNET, comme pour la fonction intégrée cos.

Pour modifier les deux programmes, appelez l'éditeur LOGO avec:

edall (Edition de la mémoire de travail)

Toutes les actions ainsi que les noms présents dans la mémoire de travail seront ainsi transmises à l'éditeur pour y être traitées. Vous trouverez par exemple des lignes de la forme suivante:

```
make "tauxtva 0.14
```

Vous pouvez, de cette manière, modifier aussi la valeur de tauxtva ou entrer ou modifier d'autres noms globaux. L'utilisation de l'éditeur dans ce but est pratique lorsque la 'ligne make' est assez longue et que vous ne voulez y apporter que de petites modifications.

Modifiez maintenant les lignes-titres des deux petites procédures pour qu'elles deviennent:

```
to TVA :prix
to PRIXNET :prix
```

Vous pouvez mettre fin à l'édition avec CONTROL+C (sur le PCW) ou COPY (sur le CPC). TVA et PRIXNET pourront ensuite être utilisés comme des fonctions mathématiques.

```
TVA 100
Réponse: 14

PRIXNET 100
Réponse: 114
```

LOGO a donc maintenant appris deux nouvelles fonctions mathématiques.

5.3 Noms globaux et noms locaux

Les valeurs numériques spéciales 156.30, 100 n'ont de signification que là où elles sont entrées, lors de l'appel de la procédure. Dans le texte du programme, elles sont utilisées sous le nom prix. Quelle serait donc, par exemple, la valeur de prix après les trois lignes indiquées plus haut?

Contrôlez cette valeur avec:

```
pr :prix
```

Mais la réponse du LOGO sera:

prix has no value (prix n'a pas de valeur)

Aucune valeur ne correspond au nom PRIX bien qu'il ait été utilisé dans les deux procédures TVA et PRIXNET. Comme les deux procédures nous renvoient des valeurs correctes, la valeur prix doit forcément exister effectivement à l'intérieur de ces procédures. Le nom prix est donc visiblement connu à l'intérieur des procédures qui prévoient prix comme valeur d'entrée mais il est inconnu en dehors de ces procédures. Ce genre de noms sont appelés noms locaux.

Il se peut que vous n'ayez pas obtenu de message d'erreur mais que LOGO ait, au contraire, sorti une valeur. Dans ce cas, cela signifierait que vous avez déjà entré, à un moment donné, au cours de la même session de travail, une ligne make "prix ... Vous pouvez le vérifier en entrant:

```
edall
```

Si vous recevez effectivement une valeur, après pr :prix, et non un message d'erreur, alors, après edall, une ligne du type make "prix ... devrait apparaître à l'écran; ces lignes apparaissent toujours à la suite des textes de programmes. Vous pouvez également accéder isolément à la ligne recherchée avec ed "prix. La ligne make "prix ... disparaîtra de la mémoire de travail après que le nom global ait été supprimé avec

ern "prix (ce qui signifie erase name "prix)

Vous pouvez vérifier que la ligne ne réapparaîtra plus après appel de PRIXNET.

Les termes global/local indiquent quand le système LOGO peut traiter les noms correspondants:

- Les noms globaux sont disponibles pour toutes les actions du LOGO.

- Les noms locaux ne valent qu'à l'intérieur d'une procédure.

Par 'toutes les actions du LOGO', nous entendons naturellement tout ce qui est entrepris, dans une session LOGO, après que le nom global ait été déclaré. Les noms globaux peuvent être aussi supprimés à nouveau. Les noms supprimés n'existent plus alors pour le LOGO.

Le fait que les noms locaux ne valent qu'à l'intérieur d'une seule procédure semble beaucoup limiter leur champ d'application. Ce n'est cependant pas du tout le cas car les procédures peuvent, à leur tour, appeler d'autres procédures. Le LOGO reconnaîtra un nom

local aussi longtemps qu'il ne sera pas parvenu à la fin de la procédure dans laquelle le nom local a été introduit.

La signification concrète de la distinction entre noms globaux et noms locaux n'apparaîtra plus nettement que plus tard, lorsque nous en viendrons à des exemples de programmes plus complexes. Pour l'instant, il faut retenir que:

Les entrées des procédures sont des noms locaux.

5.4 Sortie du vocabulaire étendu

Lorsque vous avez un doute sur le vocabulaire de base LOGO et sur sa signification, vous devez consulter une liste des mots LOGO (voir annexe). Si le vocabulaire a été complété par programmation ou par l'introduction de noms globaux, il faut qu'on puisse, à tout moment, examiner le niveau d'extension du vocabulaire auquel on est parvenu.

C'est à cela que sert le mot LOGO

po (abréviation de Print Out)

po nécessite en entrée le nom d'une procédure ou bien une liste de noms de procédures. Le texte de programme de la procédure indiquée sera alors sorti dans la fenêtre de texte. Comme pour l'appel de l'éditeur, il existe aussi des variantes:

poall (pour Print out all)

fera écrire sur l'écran toutes les procédures et noms globaux existants. Le système LOGO veille cependant à ce que l'écran ne soit pas dépassé et interrompt la sortie dès qu'une page est pleine de sorte que le texte sorti ne disparaisse pas avant que vous ayez eu le temps de le lire. La page suivante ne sera remplie qu'après que vous ayez frappé une touche.

Si vous avez une imprimante connectée à votre ordinateur, vous pouvez faire sortir sur une papier une copie de la totalité de la

mémoire de travail, avec:

```
copyon poall
```

Si vous ne suivez la sortie que sur l'écran, il est le plus souvent recommandé de limiter d'emblée la sortie avec po.

Si l'on ne s'intéresse qu'aux procédures ou qu'aux noms, on l'indique en entrant:

pops (pour Print out procedures)
pons (pour Print out names)

Lors de la sortie des noms, vous pouvez également voir à l'écran leurs valeurs respectives, par exemple:

```
tauxtva is 0.14
```

Il est également possible de sortir les procédures une à une:

```
po "TVA
po [TVA PRIXNET
```

On veut parfois simplement savoir quels nouveaux mots LOGO, c'est-à-dire quels noms de procédures, ont été déclarés. Les noms de procédures sont appelés TITLES:

pots (pour Print out titles)

On peut ainsi avoir très facilement une vue d'ensemble des actions LOGO auto-définies.

5.5 Suppression de programmes et de noms

Il arrive parfois qu'on veuille à nouveau supprimer du système certaines extensions du vocabulaire. Il est parfois préférable d'écrire un nouveau programme au lieu de corriger un programme déjà existant. Tout programmeur butera, en effet, à un moment ou à un autre, sur les limites de la mémoire. Il lui faudra alors essayer

de libérer de la place en éliminant des objets devenus superflus. Pour la suppression, on utilise le mot LOGO:

```
er
```

er nécessite une entrée, par exemple:

```
er "TVA

er "PRIXNET
```

Nous avons déjà évoqué la possibilité de supprimer des mots isolés. Il y a pour cela un mot LOGO:

```
ern "TAUXTVA
```
(pour Erase name TAUXTVA)

Vous pouvez également supprimer d'un seul coup tout le contenu de la mémoire de travail. (Faites très attention!)

erall (pour Erase All)

Cela supprimera tous les noms et toutes les procédures.

Les espaces comme signes de séparation

Vous vous êtes peut-être rendu compte que les espaces ne sont pas utilisés avec la même fréquence dans la formulation des programmes.

```
op (1+:tauxtva)*:prix
op ( 1 + :tauxtva ) * :prix
```

Ces deux lignes produiront les mêmes effets. Dans la seconde ligne, tout ce qui n'appartient pas à un même objet est séparé par des espaces. Cela correspond à la règle universelle suivante sous LOGO:

> Le caractère espace a sous LOGO une fonction de marque de séparation. Tant que des caractères d'une chaîne de caractères ne sont pas séparés par des espaces, ils sont

solidaires et constituent pour le LOGO un objet unique.

Toutefois lorsque des caractères spéciaux font partie d'une suite de caractères, la signification de ces caractères fait que le LOGO sera malgré tout capable de décomposer un bloc de caractères en plusieurs termes. Dans 1+:tauxtva, le nombre 1 et la valeur de tauxtva peuvent être aisément identifiés comme opérandes de l'addition. Le LOGO se montre, dans des circonstances semblables, très indulgent à l'égard de l'utilisateur puisqu'il reconnaît la séparation même sans le caractère espace. Vous constaterez cependant que, lors de l'édition, l'éditeur insérera de lui-même des espaces dans votre texte de programme lorsque le système le jugera nécessaire. Si des caractères spéciaux, c'est-à-dire des opérateurs arithmétiques, des parenthèses, des guillemets ou le point virgule, ne doivent pas remplir leur fonction habituelle, cela doit être indiqué en plaçant devant le caractère backslash.

Mieux vaut cependant écrire un espace de trop qu'un espace de moins que nécessaire. Toutefois, le signe moins constitue, à cet égard, un cas particulier important.

```
TVA -100
TVA - 100
```

Dans la première ligne, une valeur négative sera utilisée comme prix (peu importe ici que cela ait une signification ou non). Dans la seconde ligne, le signe moins sera, par contre, interprété comme signe d'opération parce qu'il est séparé du nombre 100 par un espace. Cela entraînera un message d'erreur.

Leçon 6: Introduction à la programmation graphique

Nouveaux éléments de vocabulaire:
fs, ts, ss, setsplit, fd, bk, rt, lt, home, window, fence, wrap, cs, clean, pu, pd, ht, st, watch, nowatch, trace, notrace

Programmes:
CARRE, ETOILE, DECALER

6.1 La fenêtre graphique

L'image du moniteur de votre ordinateur est constitué d'une grille de nombreux petits points image indépendants. Cela permet de représenter des dessins comme cela se passe avec les techniques d'impression normales même si la grille est beaucoup plus serrée dans ce dernier cas.

Nous avons utilisé l'écran jusqu'ici essentiellement pour la représentation d'opérations de calcul avec leurs résultats et pour écrire des programmes. Les zones de l'écran utilisées pour une fonction déterminée sont appelées fenêtres: la fenêtre de texte, la fenêtre d'édition et la fenêtre graphique. On entend par fenêtres des zones de l'écran bien délimitées dans lesquelles se déroulent des fonctions déterminées. Dans le système LOGO sur les ordinateurs AMSTRAD, de telles fenêtres ne peuvent se chevaucher, on peut simplement diviser l'écran.

Nous avons décrit dans la dernière leçon la fenêtre d'édition; elle comprend normalement la totalité de l'écran à l'exception d'une ligne de bas de page. Pour les dessins vous pouvez soit utiliser également la totalité de l'écran (Full screen) soit diviser l'écran en une partie supérieure consacrée au graphisme et une partie inférieure réservée à la sortie de texte (Split screen).

Les instructions suivantes permettent de définir comment l'écran devra être utilisé:

- fs (pour full screen: tout l'écran comme fenêtre graphique)
- ts (pour text screen: que la fenêtre de texte)
- ss (pour split screen: écran divisé)

C'est l'écran divisé qui sera utilisé pour le travail graphique si vous ne réclamez pas expressément fs. C'est d'ailleurs le plus pratique car cela vous permet de surveiller dans la partie inférieure de l'image du moniteur vos entrées au clavier.

La taille de la fenêtre de texte en mode split screen peut être fixée avec l'instruction setsplit.

```
setsplit 10
```

10 lignes de texte seront ainsi réservées dans la zone basse de l'écran ce qui limitera naturellement d'autant la fenêtre graphique. Il sera en général préférable de travailler tout simplement avec la taille proposée par le système.

L'instruction fs n'agit que lorsque vous videz ensuite la fenêtre graphique avec cs ou clean.

```
fs cs  ou  fs clean
```

Le mot LOGO cs (pour clear screen) place la fenêtre graphique dans sa position de départ. C'est pourquoi nous vous conseillons de commencer systématiquement tout travail graphique par

```
cs
```

Vous pouvez aussi parvenir dans la position de départ sans commencer par cs mais un léger défaut du système fait qu'une pointe de flèche indésirable demeurera fixée au centre de l'écran. Retenez donc que:

L'ouverture de la fenêtre graphique devrait en

général commencer par cs pour corriger le défaut du système indiqué. Ce sera par exemple également nécessaire à la suite d'une opération d'édition.

La programmation de graphisme informatique consiste à donner des ordres à un crayon à dessin imaginaire qui serait placé sur la fenêtre graphique. Comme vous n'avez pas ce crayon réellement en main, vous devez le commander avec des instructions. Pour se repérer sur la surface de dessin, il existe deux méthodes:

- le langage 'centré sur le corps' de ce qu'on appelle 'Turtle-graphics',

- la référence à un système de coordonnées.

Turtle graphics (Turtle = tortue) est une particularité du LOGO car c'est pour cela qu'il a été développé à l'origine. Il existe aujourd'hui d'autres environnements de langages de programmation qui prévoient cette méthode graphique car l'intérêt de Turtle graphics est maintenant bien connu.

La plupart des langages de programmation destinés à des ordinateurs à possibilités graphiques possèdent un certain nombre d'instructions permettant de dessiner dans un système de coordonnées. Le LOGO ne permet pas seulement le Turtle graphics, caractéristique du LOGO, mais il vous fournit aussi quelques aides pour une programmation graphique axée sur un système de coordonnées. Nous commencerons ici par traiter en détail de Turtle graphics. Ce n'est qu'un peu plus tard que nous nous intéresserons aux instructions qui utilisent les coordonnées.

6.2 Déplacements de la tortue

Quel rôle joue la tortue?

Lorsque vous lancez le travail graphique sous LOGO avec cs, un symbole de flèche, la tortue, apparaît au centre de la fenêtre graphique. Il s'agit d'une représentation de tortue très abstraite!

Cette petite flèche symbolise un crayon à dessin. En effet on peut dessiner directement là où se trouve la tortue. C'est volontairement que la tortue n'a pas été représentée par un point ou un petit cercle car la pointe de la tortue indique dans quelle direction se déplace le crayon à dessin.

C'est le fait de marquer la position ainsi que la direction du crayon à dessin qui caractérise le graphisme de tortue. On ne donne pas à la tortue des ordres du style 'va de tel endroit à tel autre'.

L'instruction de base qu'on donnera à la tortue sera bien plutôt:

> Avance sur une distance déterminée dans une direction déterminée!

La direction qui s'applique actuellement est indiquée par la pointe de la tortue. Essayez maintenant:

```
fd 60
```

La tortue s'est déplacée vers le haut. Elle a, en même temps, laissé sur l'écran une trace, sous la forme d'une ligne droite. Dessiner avec la tortue consiste donc à:

> déplacer la tortue sur l'écran graphique, de façon à ce que la trace qu'elle laisse sur l'écran donne le dessin voulu.

Evidemment, pour ce faire, il faut pouvoir modifier la direction de la tortue. Entrez:

```
rt 90
```

La tortue s'est effectivement tournée de 90 degrés sur la droite. Le sens de la direction, vers la 'droite', se réfère à la tortue même. Sur un véhicule, rt indiquerait le passage dans un tournant à droite.

En fait, fd et RIGHT pourraient suffir à obtenir toutes les

directions possibles car tourner de 270 degrés sur la droite revient au même tourner de 90 degrés sur la gauche. Le LOGO autorise, malgré tout, le déplacement vers l'arrière et le tournant à gauche:

```
bk 50
lt 90
```

Vous brûlez certainement d'utiliser ces instructions pour déplacer la tortue sur l'écran graphique! Allez-y et n'hésitez pas à la faire sortir un peu des limites de l'écran! Que remarquez-vous?

Lorsque la tortue dépace le bord supérieur de l'écran, par exemple avec fd 200, le symbole de flèche n'est plus visible. Vous pouvez toutefois ramener la tortue dans sa position de départ avec

home (va à la maison)

Le comportement que nous venons d'indiquer, lorsque vous tentez de faire sortir la tortue des limites de la fenêtre graphique, est celui qu'on peut observer juste après le lancement du LOGO. Mais DR LOGO connaît, à cet égard, trois modes de travail:

- window (fenêtre)
- fence (clôture)
- wrap (état boucle)

L'état window signifie que la tortue peut quitter, sans problème, la zone de la fenêtre graphique. Mais on ne verra bien sûr que ce qui se trouve à l'intérieur de la fenêtre graphique.

Avec l'état fence, la tortue butera par contre, au bord de la fenêtre graphique, sur une clôture qui l'empêchera de quitter la fenêtre. Le message d'erreur suivant apparaîtra:

```
Turtle out of bounds
```

Notez que, dans ce cas, la tortue reste sur sa position. En état

fence, une instruction ne sera alors exécutée que si elle ne fait pas sortir la tortue de la fenêtre graphique.

Les entrées faisant atteindre à la tortue le bord de la fenêtre graphique dépendent de la taille fixée pour la fenêtre de texte mais aussi du fait que vous travailliez sur le CPC ou sur le PCW. Avec le réglage standard, les bords sont atteints, en partant de la position home, avec:

- Sur le CPC Horizontalement:	192 vers le haut, 193 (112) vers le bas
	Verticalement: 320 vers la gauche ou la droite
- Sur le PCW Horizontalement:	264 vers le haut, 266 (95) vers le bas
	Verticalement: 360 vers la gauche ou la droite

En mode wrap (boucle) les bords supérieur et inférieur de la fenêtre sont solidaires, de même que les bords droit et gauche. Lorsque la tortue sort de la fenêtre graphique par en haut, elle réapparaît par en bas. De même, si elle sort par la gauche, elle réapparaît par la droite.

Essayez (vos entrées doivent éventuellement être adaptées en fonction de la taille de la fenêtre graphique):

```
home window fd 250
home fence fd 250
home wrap fd 250
home wrap rt 90 fd 400
```

Ce que nous avons indiqué au sujet des bords se réfère toutefois uniquement à la fenêtre graphique et non à l'écran.

Le mode de boucle (wrap) permet d'obtenir des effets graphiques très amusants. En effet, la tortue se déplace alors tout autour de l'écran, comme s'il s'agissait de la surface d'un pneu de vélo.

Cependant l'état window correspond peut-être mieux à ce qu'on attend d'une surface graphique.

6.2 Programmes graphiques simples

En répétant quatre fois un mouvement en avant suivi d'un tournant de 90 degrés:

```
fd 80 rt 90
```

on obtient un carré. La ligne d'entrée peut être répétée à l'aide de CONTROL+Y. Le LOGO exécute cette répétition automatiquement avec repeat, ce qui nous donnera la ligne d'entrée suivante:

```
repeat 4 [fd 80 rt 90]
```

pour dessiner un carré de 80 unités de côté.

Alors que, pour le calcul, on peut encore employer l'ordinateur utilement même sans écrire de programme, cela deviendrait vite assez pénible, pour le dessin, de devoir toujours taper et retaper les instructions destinées à la tortue. L'emploi d'un ordinateur dans le domaine du graphisme devient vraiment intéressant à partir du moment où on constitue des mouvements complexes, à partir des deux mouvements de base de la tortue, la progression linéaire et le tournant.

La première extension de langage que nous allons apprendre au LOGO dans ce domaine sera l'action 'dessiner un carré d'une longueur donnée'. Nous appellerons cette action tout simplement 'CARRE':

```
to CARRE
```

Vous pouvez également appeler l'éditeur LOGO directement, avec:

```
ed "CARRE
```

La fenêtre d'édition s'ouvrira alors avec la ligne titre de la procédure CARRE.

La nouvelle action LOGO exécutera la ligne d'entrée que nous avons déjà utilisée. Le programme apparaîtra donc ainsi en mode d'édition:

```
to CARRE


>repeat 4 [fd 100 RT 90]
```

Le programme devra être terminé par la ligne:

```
>end
```

si vous avez commencé avec to CARRE. Si vous avez utilisé l'éditeur dès le départ pour écrire ce petit programme, la procédure CARRE sera acceptée dans le vocabulaire une fois que vous l'aurez demandé avec CONTROL+C (ou avec COPY sur le CPC).

Cette procédure peut vite devenir ennuyeuse car c'est finalement toujours le même carré qui est réalisé. La tentation est donc forte de prévoir un côté variable. Il faut pour cela que la procédure CARRE prévoie que le côté soit une entrée.

```
to CARRE :cote
repeat 4 [fd :cote rt 90]
```

Essayez à cette occasion de vous représenter très clairement la signification du double point introductif avant le nom cote: :cote signifie 'valeur du côté'.

Lorsque vous serez las de dessiner des carrés de côtés différents, comme

```
CARRE 10 CARRE 20 CARRE 30
```

essayez alors d'introduire le hasard dans le graphisme. Cependant, l'écran graphique commence maintenant à être plein et il serait souhaitable de pouvoir restaurer l'état initial de la fenêtre graphique. Outre la commande prévue à cet effet, cs, que nous avons déjà utilisée en mode graphique, vous avez aussi la possibilité de

n'effacer sur l'écran graphique que les traces de la tortue même.

```
clean
```

Cela effacera le contenu de l'écran graphique sans effets secondaires alors que cs fixe un état initial bien défini du graphisme, ce qui comprend également l'effet de home.

Est-ce que le dessin suivant vous plaît?

```
repeat 100 [CARRE random 150]
```

Cette petite ligne LOGO suffit à produire le dessin de la figure 6.1, qui donne une certaine sensation de relief. Il s'agit d'une image qui, suivant le point de vue du spectateur, semble s'avancer de l'arrière ou semble au contraire plonger vers l'arrière, dans le moniteur.

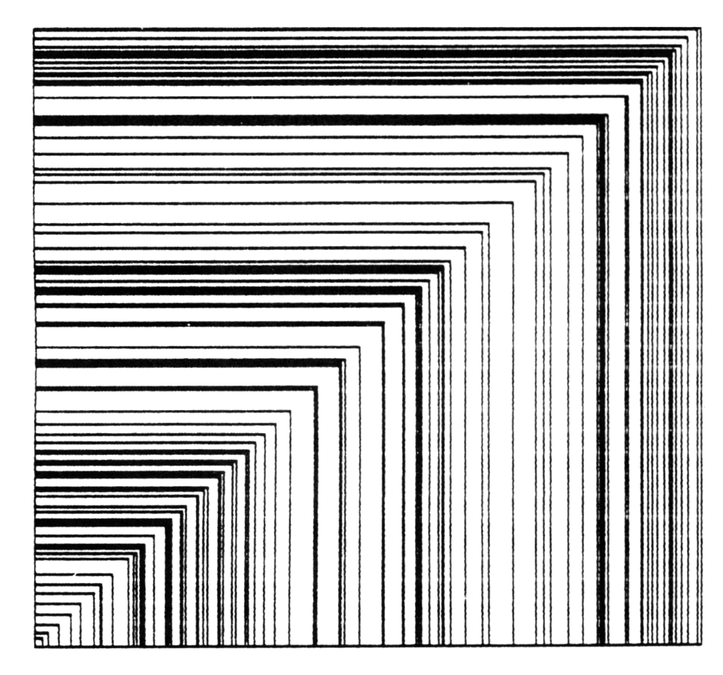


Figure 6.1: Image en relief par répétition de CARRE

Examinez encore une fois, attentivement, l'appel de l'action CARRE dans cette ligne. L'entrée du côté n'est pas faite au moyen d'un nombre mais à travers une autre action LOGO, random. Le nombre 150 est une entrée pour la procédure placée juste devant, random. random 150 fournit comme résultat un nombre unique qui sera alors employé comme côté par CARRE.

6.4 Rotation de figures

Après exécution de la procédure CARRE, la tortue a de nouveau rejoint sa position de départ, au centre de l'écran. Naturellement, la répétition de CARRE avec la même entrée produira la même figure. Si un travail quelconque est effectué, vous ne pourrez vous en rendre compte qu'à travers les mouvements de la tortue. Mais si vous déplacez la tortue avant de répéter la procédure, la figure produite lors de l'appel de CARRE ne changera pas mais sa position sur l'écran, elle, sera bien modifiée:

```
CARRE 60 rt 45 CARRE 60
```

Le second carré a subi une rotation de 45 degrés par rapport au premier. Ces figures sont cependant identiques si on les superpose (on parle de congruence des figures).

Des figures aussi simples que les carrés ne commencent à présenter un intérêt que lorsqu'on les répète plusieurs fois. C'est pourquoi nous vous demandons d'essayer la ligne suivante:

```
cs repeat 36 [CARRE 60 rt 10]
```

La facilité avec laquelle on peut ainsi faire tourner ou déplacer des figures entières constitue un des grands attraits du graphisme avec tortue.
Si le modèle créé avec la ligne

```
repeat 36 [CARRE 60 rt 10]
```

vous plaît, vous pourriez bien l'adopter comme extension du langage pour la session actuelle sous LOGO.

```
to ETOILE :cote
repeat 36 [CARRE 60 rt 10]
```

Avec une ligne telle que:

```
cs ETOILE 40 ETOILE 60 ETOILE 90
```

les modèles ainsi imbriqués peuvent produire des images déjà assez impressionnantes.

Dans la procédure ETOILE, le côté est variable mais l'angle de rotation est encore fixe, 10 degrés. Avec 36 rotations on obtient ainsi un tour complet. Le mot déclaré ETOILE nous permettrait des résultats plus variés si le nombre de rotations était modifiable.

Pour modifier la procédure ETOILE, il nous faut l'éditer. Avec:

```
ed "ETOILE
```

la fenêtre d'édition sera ouverte et le texte de ETOILE apparaîtra. Notez que le nom de procédure doit être introduit par les guillemets. Sans guillemets, vous verrez apparaître dans la fenêtre d'erreur le message:

```
Not enough inputs to ETOILE
```

DR LOGO applique ici les règles de manière stricte:

- ETOILE sans guillemets déclenche, dans tous les cas, l'exécution de l'action ETOILE. Cette action ne pourra alors être exécutée du fait de l'absence de l'entrée du côté.

- Avec les guillemets, le nom de l'action ETOILE est transmis, comme entrée, au mot LOGO ed. Si la procédure ETOILE existe, ed en sortira alors le texte dans la fenêtre d'édition, sinon seules les lignes de titre et de conclusion apparaîtront dans la fenêtre d'édition.

```
to ETOILE :cote :n
repeat :n [CARRE :cote rt 360/:n]
end
```

La transmission du texte modifié de ETOILE au système LOGO s'effectue, comme nous en avons déjà l'habitude, avec CONTROL+C ou avec COPY sur le CPC.

L'angle de rotation 360/:n fera que le résultat final sera une rotation complète. Essayez un peu

```
ETOILE 80 2 ETOILE 40 4 ETOILE 20 8
```

ou bien laissez vous surprendre par

```
ETOILE random 100 random 72
```

Avec cet appel, les deux entrées pour ETOILE seront tirées au hasard.

Pour ceux qui n'ont pas encore une grande expérience du travail sous LOGO, 360/:n peut sembler un peu étrange. La division n'est pas indiquée par le double point mais bien par le trait de fraction en diagonale. Le double point fait partie du nom n.

6.5 Déplacement de figures

Si la tortue ne tourne pas sur un certain angle après exécution de CARRE, mais si elle est déplacée sur une certaine distance, en avant ou en arrière, le carré suivant apparaîtra chaque fois décalé.

```
repeat 5 [CARRE 20 fd 20]
```

Nous plaçons ici cinq carrés à la suite les uns des autres. Pour pouvoir mieux utiliser l'écran dans les exemples suivants, nous allons commencer par déplacer la tortue vers le bas car il y aura alors plus de place au-dessus d'elle.

```
cs bk repeat 5 [CARRE 20 fd 30]
```

Les carrés ne se touchent plus cette fois. On aperçoit cependant quelque chose qui n'avait pas été prévu: les carrés décalés sont tous reliés entre eux par une ligne. Lorsque nous déplaçons la tortue avec fd 30, celle-ci laisse toujours une trace. Il serait bien sûr préférable de pouvoir l'éviter, sinon toutes les figures placées sur l'écran seraient reliées entre elles.

Pour obtenir une tortue bien propre, il faut entrer:

pu (abréviation de pen up)

pu veut dire: lever le crayon à dessin. Après cette instruction, le crayon à dessin pourra être déplacé comme auparavant mais il ne laissera plus de trace. Pour continuer le dessin, on peut l'abaisser à nouveau avec:

pd (abréviation de pen down)

Lorsqu'on veut donc effectuer un décalage, le crayon doit être relevé avant le décalage du dessin puis à nouveau rabaissé. Les mouvements du crayon à dessin peuvent être reliés de la même façon au dessin de la figure.

On peut naturellement décaler un dessin dans n'importe quelles directions et pas seulement bien sûr vers le haut. Il serait intéressant de disposer pour cela d'un mot LOGO particulier:

```
to DECALER :angle :distance
ht pu rt :angle
fd :distance
lt :angle pd st
end
```

La direction est indiquée ici par l'angle par rapport à la verticale dirigée vers le haut. Sur une carte d'état-major, l'Est correspondrait par exemple à l'angle +90 degrés, l'Ouest à l'angle -90 degrés, le Sud-Ouest à l'angle -135 degrés, etc... Dans la procédure DECALER, la rotation sur la droite doit être à nouveau

annulée, après le déplacement en avant, par une rotation équivalente sur la gauche. Sinon, en effet, on aurait, en plus du décalage voulu, une rotation de la figure qui n'était pas dans nos intentions.

Essayez par exemple la ligne suivante:

```
cs repeat 5 [CARRE 20 DECALER -45 35]
```

Vous avez certainement remarqué, dans le programme DECALER, les abrévations non-expliquées ht et st. Le dessin des cinq carrés permettait à peine toutefois de remarquer leur effet.

```
ht (Hide Turtle, cacher la tortue)
st (Show Turtle, montrer la tortue)
```

Ces instructions permettent de rendre la tortue invisible ou visible. Il n'est pas conseillé, en général, de cacher la tortue pendant le travail de dessin car les mouvements de la tortue montrent nettement les modifications apportées dans l'écran graphique. Mais si on est gêné par la présence de la tortue dans le dessin terminé, on peut toujours la cacher à la fin du dessin, avec ht.

Dans de longs programmes de dessin, le fait de cacher la tortue a l'avantage d'accélérer l'opération de dessin. Il est facile de comprendre que cela prend également du temps à l'ordinateur de représenter la tortue en mouvement.

6.6 Les procédures appellent des procédures

Dans les programmes graphiques élémentaires que nous avons examinées jusqu'ici, c'est le dessin sur l'écran qui jouait le rôle principal. Les procédures que nous avons ainsi créées sont pour partie plus complexes que les programmes tout simples de calcul de la TVA. Pour bien montrer la structure d'un programme, nous allons reprendre pour l'examiner à la loupe la dernière version du programme ETOILE. Il nous faut aussi pour cela la procédure CARRE.

```
to ETOILE :cote :n
repeat :n [CARRE :cote rt 360/:n]
end
to CARRE :cote
repeat 4 [fd :cote rt 90]
end
```

Dans la procédure ETOILE, est utilisé le mot LOGO, défini par l'utilisateur, CARRE alors que le programme CARRE n'utilise, au contraire, que des actions faisant partie du vocabulaire de base du LOGO.

Lorsque de nouveaux mots LOGO ont été créés par programmation, ils sont traités de la même façon que les mots de base du LOGO ('LOGO primitives'). Après appel de l'action CARRE, cette procédure enclenche l'action LOGO fd. L'important est à cet égard que l'entrée pour la procédure fd vienne de celle du programme qui appelle CARRE.

Si le programme ETOILE est appelé avec

```
ETOILE 50 4
```

le nombre 50 sera d'abord transmis sous le nom cote à la procédure ETOILE. A l'intérieur de ETOILE, le nom cote sera transmis alors à la procédure CARRE, puis, dans le programme CARRE, à l'action de base LOGO fd. Ce n'est qu'à cet endroit que le nombre 50 sera employé directement pour une action sur l'écran.

- les procédures peuvent à leur tour appeler d'autres procédures

- les valeurs en entrée peuvent être transmises, avec leurs noms, aux procédures appelées.

La possibilité de transmettre des valeurs en entrée de procédures à d'autres procédures appelées par ces première procédures facilite une structuration modulaire des programmes. Pour bien utiliser une procédure, il suffit de savoir quelles valeurs sont nécessaires en entrée, et dans quel ordre. Peu importe le nom que ces valeurs en

entrée reçoivent dans le programme ainsi que ce qui se passe vraiment dans ce programme. Pour autant que le programme fournisse aussi des résultats qui devront être traités, il en va de même pour les sorties.

Une fois que l'action ETOILE a été programmée une fois et testée avec succès, elle peut être intégrée dans des programmes graphiques plus complexes sans qu'on ait encore besoin de se préoccuper du programme ETOILE lui-même. Ce mode de transmission de valeurs en entrée ne fonctionne cependant sans poser de problème que parce que les entrées des procédures sont des noms locaux (ou des noms privés si vous préférez). Cela apparaît encore plus clairement dans la version suivante:

```
to ETOILE :longueur :n
repeat :n [CARRE :longueur rt 360/:n]
end
to CARRE :cote
repeat 4 [fd :cote rt 90]
end
```

A l'intérieur de la procédure ETOILE, l'entrée est appelée longueur mais là où elle est effectivement employée, dans la procédure CARRE, elle s'appelle par contre cote. On peut expliquer cela de la façon suivante:

Lorsque fd est traité dans la procédure CARRE, il faut une valeur en entrée; DR LOGO trouve à cet endroit la demande de regarder la valeur de cote. Mais cote est une entrée de CARRE. Le LOGO doit donc voir ce qui a été transmis comme valeur d'entrée pour CARRE à l'intérieur de la procédure d'appel ETOILE. Mais l'entrée a maintenant, là-bas, la 'valeur de longueur'. longueur est à son tour une valeur en entrée de ETOILE, le LOGO doit donc rechercher la valeur en entrée effective de la procédure ETOILE.

L'action fd ne reçoit donc sa valeur en entrée qu'après plusieurs détours!

Dans la transmission d'entrées à une procédure que nous avons décrite ici, c'est toujours la valeur et pas le nom correspondant

qui est transmis. C'est pourquoi on parle d'appel de la valeur (en anglais: call by value). Si vous connaissez des langages de programmation tels que le PASCAL, ALGOL ou PLI, vous allez peut-être demander ce qu'il en est de ce qu'on appelle les appels de référence (en anglais: call by name). Ne vous inquiétez pas, le LOGO permet aussi cela. Nous nous y intéresserons lorsque nous traiterons du mot LOGO thing.

6.7 Watch et trace: les contrôleurs de DR LOGO

L'imbrication d'appels de procédures permet d'obtenir des actions puissantes. Il faut cependant toujours faire attention pour ne pas perdre la maîtrise des opérations lorsque la complexité des procédures devient trop grande. DR LOGO vous offre deux moyens de suivre pas à pas le déroulement d'un programme. Entrez pour cela:

trace (suivre le déroulement de procédures)

Ce mode persiste jusqu'à ce que vous entriez:

notrace (fin du suivi)

Dans la recherche des erreurs, il peut être extrêmement utile de combiner ce mode avec la sortie sur imprimante, si vous en possédez une:

```
copyon trace
```

Faites maintenant un essai, avec

```
ETOILE 50 2
```

Une suite de messages apparaît:

```
[1] Evaluating ETOILE
[1] n is 2
[1] longueur is 50
[2] Evaluating CARRE
[2] cote is 50
```

```
[2] Evaluating CARRE
[2] cote is 50
```

En mode TRACE, l'exécution des procédures est retracée par des messages sur l'écran. Evaluating ... signifie qu'on a maintenant commencé la procédure ... Les valeurs en entrée sont ensuite révélées, avec leur nom et la valeur transmise.

Observez l'effet de TRACE dans la procédure

```
to TVA :prix
MAKE "TAUXTVA .14
op :tauxtva* :prix
end
```

MAKE est ici utilisé pour affecter une valeur à un nom, ce qui entraîne la ligne trace:

```
[1] Making "TAUXTVA 0.14
```

TVA produit aussi une sortie qui sera indiquée, en mode trace, par le commentaire:

```
[1] TVA returns ...
```

Que veulent dire les nombres sortis entre crochets au début des lignes de messages?

Dans l'exemple ETOILE, la procédure ETOILE est lancée directement lors de l'appel ETOILE 50 2 alors que CARRE ne sera appelé qu'à travers ETOILE. CARRE est donc utilisé à un autre niveau que ETOILE, ce qui est indiqué par les chiffres placés entre crochets.

Cela apparaît d'autant plus nettement lorsqu'on a encore plus recours à l'imbrication des appels de procédures. Si la ligne de messages commence par [3], la procédure indiquée a été appelée par une autre action qui avait elle-même été lancée par une troisième, etc...

Le mode trace n'est pas utile uniquement pour la recherche des

erreurs car il permet aussi d'apprendre à comprendre le déroulement parfois compliqué, imbriqué des programmes LOGO. C'est justement parce que le LOGO est un langage de programmation puissant que la logique interne d'un programme LOGO peut aussi devenir beaucoup plus compliquée que cela ne serait possible dans des langages de programmation plus primaires.

Outre le mode trace qui vous permet de suivre le déroulement du programme, vous disposez encore d'une autre possibilité de contrôle, le mode Watch.

Essayez

```
watch
ETOILE 50 2

[1] In ETOILE, repeat :n [CARRE :longueur rt 360 / :n]
```

Il faut que vous appuyiez chaque fois sur la touche RETURN pour que cela continue.

```
[2] In CARRE, repeat 4 [fd :cote rt 90]
[2] In CARRE, repeat 4 [fd :cote rt 90]
```

Ici aussi, des messages vous permettent de suivre le déroulement du programme! Les chiffres entre crochets, au début de la ligne de message, ont la même signification qu'avec TRACE. Toutefois, l'ordre des appels de procédures et les modifications éventuelles de valeurs, qui vous étaient indiquées en mode trace, ne sont pas annoncées ici. On vous montre plutôt la suite de lignes de programmes exécutées. La procédure actuellement exécutée par DR LOGO et le niveau d'imbrication sont indiquées au début de la ligne de message.

Il est mis fin à l'état Watch avec:

```
nowatch
```

Vous obtenez bien sûr les renseignements les plus détaillés lorsque vous utilisez aussi bien la fonction Watch que la fonction Trace.

Si vous avez appelé watch avec l'appel ETOILE 50 2 dans la même ligne d'entrée, vous avez obtenu un message d'erreur car watch peut également recevoir des entrées.

```
watch "ETOILE
```

Cette ligne d'instruction aurait pour effet de limiter les messages au traitement de la procédure ETOILE. On peut également demander à watch de surveiller plusieurs procédures en même temps. Il faut pour cela former une liste mais nous y reviendrons plus tard.

Il est très utile pour la recherche des erreurs de pouvoir limiter l'état Watch à quelques procédures car les informations sorties peuvent ainsi être circonscrites à l'environnement d'une erreur apparue.

Exercices

1) La tortue doit se déplacer au hasard sur l'écran. La direction et la distance parcourue doivent être déterminées avec la fonction random. La valeur en entrée pour le programme sera le nombre de répétitions.

2) Ecrivez des procédures dessinant des rectangles et des triangles de dimensions variables.

3) L'écran doit être recouvert au moyen d'un modèle de carrés de taille identique qui seront apposés les uns aux autres, sans interstices.

4) Formulez un action LOGO MAISON qui dessinera simplement une maison.

5) Produisez plusieurs maisons par déplacement d'une maison.

6) Faites apparaître de nombreuses étoiles sur l'écran, réparties au hasard.

Leçon 7: Les programmes graphiques avec répétition

Nouveaux éléments de vocabulaire :
local, if, stop, récursion, comparaisons

Programmes :
PLUSIEURSCOTES, Programmes pour cercles, arcs de cercles, ROSETTE, POLYGONE, BRODER (courbes désordonnées), COINS, LUTINS, DOUBLESCOINS

7.1 Du carré au cercle

Après exécution de la procédure CARRE

```
to CARRE :cote
repeat 4 [fd :cote rt 90]
end
```

la tortue se retrouve sur sa position de départ. C'est bien le but du programme mais il ne va pas de soi, au fond, que la répétition, par quatre fois, d'une instruction graphique produise une figure close. Comment peut-on voir cela d'après le programme?

Cela vient d'abord des mouvements de rotation. Une rotation, à quatre reprises, de 90 degrés donne une rotation de 360 degrés, donc un tour complet. C'est pourquoi la tortue se retrouve orientée, à la fin, dans la même direction verticale qu'au début. Il pourrait cependant encore arriver qu'on n'obtienne pas une figure fermée parce que la tortue aurait pu atteindre, à la fin, un autre point que son point de départ.

Il faut donc qu'elle se déplace vers le haut sur la même distance que vers le bas; il en va de même pour ses mouvements sur la droite

ou sur la gauche. Tous les trajets que la tortue parcourt ont la même longueur, de sorte que, avec un angle de rotation global de 360 degrés, on aboutit à un carré.

Un tour complet de la tortue sur elle-même peut bien sûr être aussi obtenu avec un autre nombre de rotations. Essayons donc un programme PLUSIEURSCOTES qui sera une généralisation de CARRE:

```
to PLUSIEURSCOTES :cote :n
repeat :n [fd :cote rt 360/:n]
end
```

Amusez-vous un peu à tester ce nouveau mot, PLUSIEURSCOTES!

```
PLUSIEURSCOTES 60 3
PLUSIEURSCOTES 60 5
PLUSIEURSCOTES 60 6
...
```

On obtient bien des triangles, des carrés, des pentagones, des hexagones, etc... réguliers!

Nous pourrions nous livrer à cet endroit à d'intéressantes réflexions sur les polygones réguliers. Contentons-nous d'examiner encore une fois le triangle régulier, c'est-à-dire le triangle à côtés égaux:

```
PLUSIEURSCOTES 80 3
```

La procédure PLUSIEURSCOTES effectue une rotation sur un angle de 360/3=120 degrés. En géométrie, on appellera cet angle de rotation de la tortue l'angle externe. L'angle interne est fourni chaque fois par 180 degrés moins l'angle externe, soit 60 degrés. On en déduit la règle bien connue selon laquelle la somme des angles internes d'un triangle est toujours de 180 degrés.

Pour voir les effets de la procédure PLUSIEURSCOTES avec un nombre croissant d'angles, on peut utiliser le petit programme suivant:

```
to PLUSIEURS.PLUSIEURSCOTES :cote :nmax
cs pu lt 90 fd 100 rt 90 pd
local "k make "k 2
repeat :nmax -1 [PLUSIEURSCOTES :cote :k make "k :k+1]
```

Essayez par exemple:

```
PLUSIEURS.PLUSIEURSCOTES 18 40
```

Vous devriez maintenant voir apparaître à l'écran une image cônique. Ce programme contient des éléments de langage que nous n'avons pas encore utilisés jusqu'ici.

Cette procédure a été appelée PLUSIEURS.PLUSIEURSCOTES. La longueur de ce nom n'est pas très pratique pour l'appel de la procédure mais, d'un autre côté, les noms longs présentent l'avantage de s'expliquer d'eux-mêmes. 'Plusieurs plusieurscôtés' sont deux mots qui, en réalité, ne devraient pas se tenir. Or l'espace joue systématiquement, sous LOGO, le rôle de marque de séparation, comme vous le savez bien maintenant. Dans la ligne

```
PLUSIEURS PLUSIEURSCOTES :cote :nmax
```

le nom de procédure serait PLUSIEURS. PLUSIEURSCOTES serait donc considéré comme une entrée.

Le point, dans PLUSIEURS.PLUSIEURSCOTES, sert de symbole de séparation optique alors qu'il permet, pour le LOGO, de ne faire qu'un mot à partir de deux. L'utilisation du point comme signe de non-séparation n'existe pas qu'en LOGO. On utilise souvent aussi, dans le même but, le caractère de soulignage que vous trouvez sur le CPC au-dessus du zéro et sur le PCW au-dessus du trait d'union, dans la plus haute rangée du clavier.

La seconde ligne de PLUSIEURS.PLUSIEURSCOTES a pour effet de vider l'écran graphique, le crayon à dessin étant déplacé vers la gauche pour qu'il reste suffisamment de place pour les polygones. La troisième ligne contient un nouveau mot LOGO, local, sur lequel nous reviendrons spécialement dans la prochaine section.

Dans la quatrième ligne, le dessin de polygones est répété (nmax-1) fois, avec REPEAT. nmax représente le nombre maximum de côtés avec lequel PLUSIEURSCOTES peut être appelé. Comme nous commençons avec un bi-angle, la répétition devra simplement être exécutée une fois de moins.

Pour que le nombre de côtés augmente lors du prochain appel de PLUSIEURSCOTES, la valeur du nombre actuel de côtés, k, sera augmentée de 1 avec make "k :k+1.

On dessinera ainsi, successivement, des polygones de 2, 3, 4, ... jusqu'à nmax côtés. Les côtés auront cependant toujours la même longueur. Le nombre de polygones s'accroîtra sans cesse, en même temps que le nombre de côtés.

Il vous est certainement déjà apparu que, du fait de la résolution limitée de l'écran graphique, certains traits ne sont pas totalement droits et on dirait que la règle de l'ordinateur présente des irrégularités. Dès qu'on en arrive au polygone à vingt côtés, il est déjà à peine possible de discerner précisément les 20 côtés. Si la règle de l'ordinateur est irrégulière, il en va bien sûr de même pour les cercles tracés par l'ordinateur. A cause de la résolution limitée de l'écran graphique, les polygones réguliers avec un grand nombre de côtés ne peuvent plus être distingués des cercles: d'ailleurs, en réalité, les cercles sont produits sur l'ordinateur à travers des polygones réguliers!

7.2 Programmation structurée avec les noms locaux

Examinons maintenant la signification de la troisième ligne du programme PLUSIEURS.PLUSIEURSCOTES:

```
local "k make "k 2
```

Le sens de make "k 2 ne pose pas de problème. Le nom k est doté d'une valeur de départ de 2. La procédure peut être également écrite sans local "k et elle produirait le même résultat. Vous pourrez constater l'effet de local "k si vous entrez pons après un appel de PLUSIEURS.PLUSIEURSCOTES. Vous chercherez en vain

le nom k. Si vous laissez par contre tomber local "k, vous trouverez, après pons, le nom k avec la valeur correspondante.

> local fait du nom qui suit un nom local.

Le nom k ne sera connu qu'à l'intérieur de la procédure dans laquelle il a été déclaré avec local. Une fois que le LOGO aura terminé cette procédure, le nom ne sera plus disponible et il sera traité exactement comme une entrée de procédure. La différence est simplement qu'une entrée de procédure reçoit une valeur lors de l'appel de la procédure alors que le nom introduit par local n'est doté d'une valeur que lorsque l'instruction make est utilisée.

Quel est donc l'intérêt de l'introduction de noms locaux avec local?

Dans l'exemple de la procédure PLUSIEURS.PLUSIEURSCOTES, il importe assez peu que k soit traité comme un nom local ou comme un nom global. Il est cependant conseillé d'utiliser des noms locaux partout où l'emploi de noms globaux n'est pas indispensable. La validité des noms locaux est limitée dès le départ à la zone où ces noms sont nécessaires. Il ne peut y avoir de ce fait de conflit ou de confusion imprévue avec des noms utilisés en dehors de cette zone. Le nom k peut être utilisé en dehors de la procédure PLUSIEURS.PLUSIEURSCOTES, l'utilisation du même nom pour un nom local ne posera aucun problème.

Lorsqu'on utilise par contre des noms globaux, on doit faire très attention à ce que n'apparaissent pas des conflits sur la signification des noms.

> L'emploi de noms locaux favorise la programmation modulaire, structurée.

Pour utiliser une procédure toute faite, il suffit alors de connaître les entrées et, le cas échéant, la sortie.

Les noms globaux qui ne sont plus utilisés constituent aussi une sorte de poubelle de programmation qui consomme de la place en mémoire. Avec les petits exemples de programmes que nous avons vus

jusqu'ici, il n'y a bien sûr pas de problème de place mémoire disponible dans l'ordinateur. Dans la pratique, l'économie de place mémoire constitue cependant un autre avantage important des noms locaux.

7.3 Tours et détours de la tortue

Cercles

Si un cercle peut être représenté, de manière approximative, au moyen de polygones, ce sont la longueur d'un côté et le nombre de côtés n du polygone que l'on peut fixer. La longueur d'un côté multipliée par n permet alors d'obtenir le périmètre du polygone. Il n'est pas habituel, cependant, de définir un cercle par son périmètre. C'est plutôt par son rayon qu'on définira un cercle. Pour pouvoir donc produire un cercle de rayon donné sous la forme d'un polygone, il faut calculer à partir du rayon le côté du polygone utilisé comme approximation du cercle. Comme le périmètre du cercle de rayon r est fourni par 2*pi*r, la longueur de côté nécessaire sera calculée à partir de:

Longueur de côté = 2*pi*r/n

Nous pouvons maintenant formuler un programme de cercles non sans avoir mis à disposition du système la constante pi, avant l'appel de notre programme:

```
to CERCLE.1 :r :n
PLUSIEURSCOTES 2*:pi*:r/:n :n
end
to PLUSIEURSCOTES :cote :n
repeat :n [fd :cote rt 360/:n]
end
```

CERCLE.1 désignera la première version d'un programme de cercles. Comme le nombre d'angles est encore variable, puisque fourni comme valeur en entrée, on peut faire des essais avec ce programme:

```
make "pi 3.141593
```

```
CERCLE.1 70 360
CERCLE.1 70 72
CERCLE.1 70 36
```

Avec 360 angles, la tortue a besoin de pas mal de temps. 36 côtés suffisent cependant déjà à produire une impression de ligne de cercle suffisamment ronde. Une seconde version utilisera 36 côtés et se distinguera de la première par le fait que l'emplacement du crayon à dessin lors de l'appel sera pris comme centre du cercle.

```
to CERCLE.2 :r
pu fd :r rt 95 pd
PLUSIEURSCOTES :pi*:r/18 36
pu lt 95 bk :r pd
end
```

Dans la seconde ligne, la tortue est d'abord amenée à partir du centre vers un point de la ligne du cercle situé à la verticale du centre. La tortue tourne alors de 95 degrés dans la direction de la ligne de cercle. Par rapport à la première version, on effectue une rotation initiale supplémentaire de 5 degrés. (L'angle supplémentaire représente exactement la moitié de l'angle de 10 degrés qui sera ensuite utilisé par PLUSIEURSCOTES.) Sans cette rotation initiale supplémentaire, le cercle serait un peu décalé sur la gauche.

Vous pouvez vous convaincre vous-même, par des essais, du fait que, sans cet angle supplémentaire, la tortue ne se retrouverait pas finalement au centre du cercle comme nous le voulons. Le déplacement en avant, du point de départ à la ligne du cercle, doit se faire sur un axe de symétrie du polygone. Comme dans le programme PLUSIEURSCOTES on commence par un déplacement en avant, la rotation supplémentaire de 5 degrés est nécessaire et à la fin on atteint une rotation totale de 360 degrés avec lt 95.

Comme la procédure PLUSIEURSCOTES ne contient qu'une seule ligne repeat, nous pouvons bien la reprendre dans la procédure du CERCLE. La version suivante, CERCLE.3, pourra alors fonctionner de façon indépendante.

```
to CERCLE.3 :r
pu fd :r rt 95 pd
local "cote make "cote :pi*:r/18
repeat 36 [fd :cote rt 10]
pu lt 95 bk :r pd
end
```

Nous avons introduit ici le nom cote pour qu'il ne soit pas nécessaire de calculer le nombre :pi*R/18 à chaque étape.

Arcs de cercle

Pour des programmes de dessin, les arcs de cercle, c'est-à-dire des sections de la ligne du cercle, sont encore plus intéressants que des cercles entiers. Ils peuvent en effet être utilisés comme standards pour des parties courbes d'un dessin. A partir d'un programme de cercle, il est bien sûr assez facile de réaliser un programme d'arcs de cercles. On peut alors dessiner des lignes incurvées vers la droite ou vers la gauche.

La position actuelle de la tortue définit le début de la courbe et il faudra donc deux actions différentes pour les incurvations vers la droite ou vers la gauche. Nous ne décrirons ici que la courbe vers la droite car vous pouvez effectuer facilement la traduction en courbes vers la gauche en convertissant toutes les rotations sur la droite en rotation sur la gauche.

Si pour accélerer l'exécution nous choisissons pour chaque étape un angle de rotation assez élevé, de 10 ou même 5 degrés, cela pose un petit problème: en effet l'angle total de rotation souhaité pour l'arc de cercle ne sera pas toujours nécessairement un multiple entier de l'angle minimum utilisé pour chaque rotation. Par exemple, pour un angle total de 86 degrés, nous ne pourrions dessiner, en suivant la méthode utilisée dans le programme de cercle, qu'un angle de 80 degrés.

Pour les 6 degrés restants, il nous faut prévoir une partie spéciale dans le programme. La subdivision de l'angle en 86 = 80+6 degrés se fera avec les fonctions de nombres entiers quotient et

remainder:

(quotient :angle 10)
(remainder :angle 10)

La première ligne fournit le nombre de pas de 10 degrés, la seconde ligne l'angle restant. Les parenthèses ne sont pas indispensables ici. Elles servent uniquement à réunir trois objets pour la présentation. Notez qu'on obtient chaque fois un seul nombre. Tout ce qui est entre parenthèses représente un nombre, ici par exemple le nombre de répétitions.

Pour déterminer l'incurvation de la courbe, on peut utiliser soit le rayon du cercle correspondant, soit la longueur du côté du polygone correspondant. La version suivante utilise directement la longueur du côté.

```
to COURBEDROITE :cote :angle
rt 5
repeat (quotient :angle 10) [fd :cote rt 10]
RESTECOURBEDR :cote/10 (remainder :angle 10)
end
to RESTECOURBEDR :cote :angle
fd :cote*:angle
rt :angle-5
end
```

Comme plus haut, dans le programme CERCLE.3, si le reste de la courbe vers la droite est produit dans la procédure COURBEDROITE, on peut renoncer à la seconde procédure. L'appel d'une procédure particulière, RESTECOURBEDR est la solution la plus élégante parce que le fait de transmettre remainder :angle 10 sous la forme de l'entrée d'une procédure fait que cette fonction ne devra être évaluée qu'une seule fois. Au contraire, lorsque nous avons intégré la procédure PLUSIEURSCOTES dans l'exemple CERCLE.3, nous avons introduit un nom local particulier auquel une valeur a été affectée avec make. Ce type de solutions est moins élégant et peu représentatif du style des programmes LOGO.

COURBEDROITE permet aussi de dessiner des cercles, il suffit

simplement que l'angle soit alors de 360 degrés.

```
COURBEDROITE :pi*70/18 360
```

Cette ligne produit un cercle d'un rayon de 70 unités.

Figures avec arcs de cercle

A l'aide des courbes vers la droite et vers la gauche, on peut produire beaucoup de très belles figures. Pour vous en donner une petite idée:

```
to SECTION :taille :angle
repeat 2 [COURBEDROITE :taille :angle rt 180-:angle]
end

to ROSETTE :taille :angle :n
repeat :n [SECTION :taille :angle rt 360/:n]
CERCLE.3 :taille ht
end
```

Essayez par exemple:

```
fs cs ht ROSETTE 15 80 9
```

ou

```
fs cs ht ROSETTE 17 80 9 ROSETTE 25 40 18
```

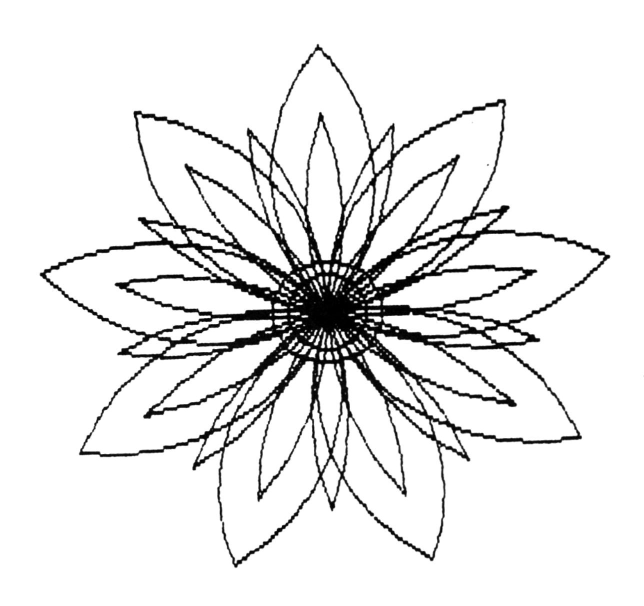


Figure 7.1: Rosettes

La combinaison avec des courbes sur la gauche offre bien sûr des possibilités encore plus amusantes de jeux de graphisme!

7.4 Répétition par récursion

Les programmes graphiques que nous avons vus jusqu'ici répondaient tous à un schéma très simple: une étape de base constituée par un déplacement en avant suivi d'une rotation sur la droite. Puis répétition de l'étape de base.

```
fd :cote rt :angle ;ETAPE DE BASE
```

Ce qui suit le point-virgule n'est pas un nouveau mot LOGO. Etape de base constitue plutôt une explication du contenu de la ligne. Il n'est généralement pas nécessaire de commenter les programmes très simples et très courts pourvu qu'on utilise dans le programme des noms aussi parlants que possible, c'est-à-dire des noms qui

s'expliquent d'eux-mêmes.

Il peut cependant facilement arriver que la signification d'un programme ne s'éclaire qu'après de longues réflexions ou de nombreux tests. C'est là que les commentaires dans le texte du programme peuvent se révéler très utiles. Par rapport à d'autres formes d'explications d'un programme (notice par exemple), les commentaires dans le programme ont l'avantage d'être présents en permanence pendant qu'on travaille sur un programme.

Le début d'un commentaire est marqué par le point-virgule. Le commentaire se termine avec la fin de la ligne. La fonction du point-virgule est donc que, sous LOGO, tout ce qui se trouve placé sur une ligne après le point-virgule sera ignoré par le système LOGO. Cela veut dire que le LOGO ne cherchera pas à traiter comme des instructions le texte suivant un point-virgule et qu'il passera directement à la ligne suivante du programme. Des points-virgules figurant par erreur dans le texte du programme entraîneront donc des erreurs dans l'exécution.

La répétition de l'étape de base se faisait jusqu'ici avec repeat:

```
repeat 1000 [fd :cote rt :angle]
```

Mais la répétition peut aussi se faire de la manière suivante:

```
to POLYGONE :cote :angle
fd :cote rt :angle ;ETAPE DE BASE
POLYGONE :cote :angle ;APPEL RECURSIF
end
```

Dans la quatrième ligne, on appelle, à l'intérieur de l'action POLYGONE, cette même procédure. C'est ce qu'on appelle la récursion.

Les programmes ne peuvent donc pas uniquement appeler d'autres programmes, ils peuvent aussi s'appeler eux-mêmes! Cela semble un peu curieux de prime abord mais vous comprendrez vite les effets que cela permet d'obtenir.

Essayez tout d'abord de voir si le programme POLYGONE peut vraiment fonctionner:

```
POLYGONE 70 60
```

Nous obtenons un hexagone régulier. Mais la tortue semble être devenue folle car elle ne cesse pas de courir tout autour de l'hexagone sans vouloir s'arrêter. En tout cas, l'auto-appel a bien produit la répétition souhaitée.

Le programme peut être interrompu de temps en temps avec CONTROL+Z. Pendant la pause, vous pouvez intervenir dans le déroulement du programme et même dans le texte du programme. co vous permet de faire se poursuivre l'exécution du programme.

Il devient vite ennuyeux de suivre la tortue se démenant autour de l'hexagone. Vous pouvez alors interrompre le programme définitivement avec ESC (STOP sur le PCW).

Cet exemple simple illustre l'effet d'un appel récursif: POLYGONE demande au système LOGO de lancer de nouveau la procédure qui demande au LOGO de lancer de nouveau la procédure qui ... etc... Un processus sans fin. Ce mode simple de récursion ne peut en fait être employé utilement que pour des répétitions dont on ne veut pas fixer le nombre par avance et lorsqu'on a choisi volontairement de faire interrompre l'exécution par une intervention extérieure.

7.5 Récursion avec entrées modifiées

Les choses se présentent très différemment dès lors qu'une procédure s'appelle elle-même mais en modifiant les entrées pour la procédure.

```
to POLYGONE.2 :cote :angle
fd :cote rt :angle ;ETAPE DE BASE
POLYGONE.2 :cote+2 :angle ;APPEL RECURSIF
end
```

La seule modification apportée à la version précédente c'est que

lors de l'auto-appel, la première entrée ne sera plus :cote mais :cote+2.

Vous devez entrer:

```
ed "POLYGONE
```

Dans la fenêtre d'édition apparaît le texte de la procédure POLYGONE déjà définie. Ajoutez maintenant ".2" à POLYGONE (dans les première et avant-dernière lignes) et écrivez encore "+2" à la suite de cote. Terminez par COPY ou ALT+C sur le PCW et POLYGONE.2 sera défini, l'ancienne procédure POLYGONE restant intacte.

Faites tranquillement quelques essais avec ce programme avant que nous n'en étudions la structure de plus près. Vous pouvez entrer par exemple:

```
cs POLYGONE.2 1 90
cs POLYGONE.2 1 60
cs POLYGONE.2 1 45
```

Les figures ainsi créées sortent vite du cadre de l'écran ce qui peut d'ailleurs donner des effets très intéressants en mode de boucle. Au lieu d'interrompre l'exécution du programme manuellement, avec CONTROL+G, on peut obtenir ici aussi une interruption avec fence, dès qu'on commandera au crayon à dessin de sortir du cadre de l'écran.

```
cs fence POLYGONE.2 1 120
```

CONTROL+Y vous permet de répéter cette ligne d'entrée autant de fois que vous le voulez. Vous n'avez alors plus qu'à modifier l'angle. S'il y a trop de lignes qui se croisent, on peut modifier la seconde ligne de façon à ce que le côté grandisse plus vite. Par exemple: :cote+4.

Essayez maintenant de voir un peu ce qui se passe avec des angles qui ne sont pas des quotients ou multiples entiers de 360, par exemple:

```
cs fence POLYGONE.2 1 49
```

Essayez de deviner l'effet que produira l'appel suivant:

```
cs fence POLYGONE.2 1 91
```

Vous pouvez peut-être deviner plus facilement, maintenant, la figure que vous obtiendrez avec un angle de 89 degrés!

Ce qui est stupéfiant dans de tels programmes, c'est la diversité des figures obtenues simplement par une modification de l'angle. De telles figures peuvent aussi être obtenues sans récursion mais la récursion offre ici des possibilités particulièrement élégantes.

Revenons-en maintenant à l'appel récursif. Pour bien comprendre comment fonctionne un programme récursif, on peut tirer profit de l'option trace. Activez donc le mode de description du programme, avec trace, avant d'appeler le programme. Si vous avez une imprimante, activez-la également, avec copyon.

```
cs trace fence POLYGONE.2 20 90
```

Vous voyez alors apparaître sur l'écran la description suivante des appels de procédures:

```
[1] Evaluating POLYGONE.2
[1] angle is 90
[1] cote is 20
[2] Evaluating POLYGONE.2
[2] angle is 90
[2] cote is 22
[3] Evaluating POLYGONE.2
[3] Evaluating POLYGONE.2
[3] angle is 90
[3] cote is 24
...
```

Lors de chaque appel de procédure apparaît d'abord, en mode trace, une ligne qui indique l'exécution de procédures déterminées. Sont ensuite sorties les valeurs actuelles des entrées. La description

donnée par le système nous montre que la récursion fonctionne ici comme si la même procédure était constamment appelée, chaque fois avec des valeurs en entrée nouvelles, modifiées.

Il en résulte que ce mode élémentaire de récursion peut aussi être remplacé par une autre structure de programmation:

```
to POLYGONE.3 :cote :angle :n;sans récursion
repeat :n [fd :cote rt :angle make "cote :cote+1]
end
```

Le nombre de répétitions doit être indiqué pour repeat. On pourrait pratiquement obtenir le même effet de fonctionnement sans fin en utilisant ici une valeur très élevée pour n:

```
POLYGONE.3 10 90 10000
```

Les répétitions d'actions sans récursion sont aussi appelées itération d'actions.

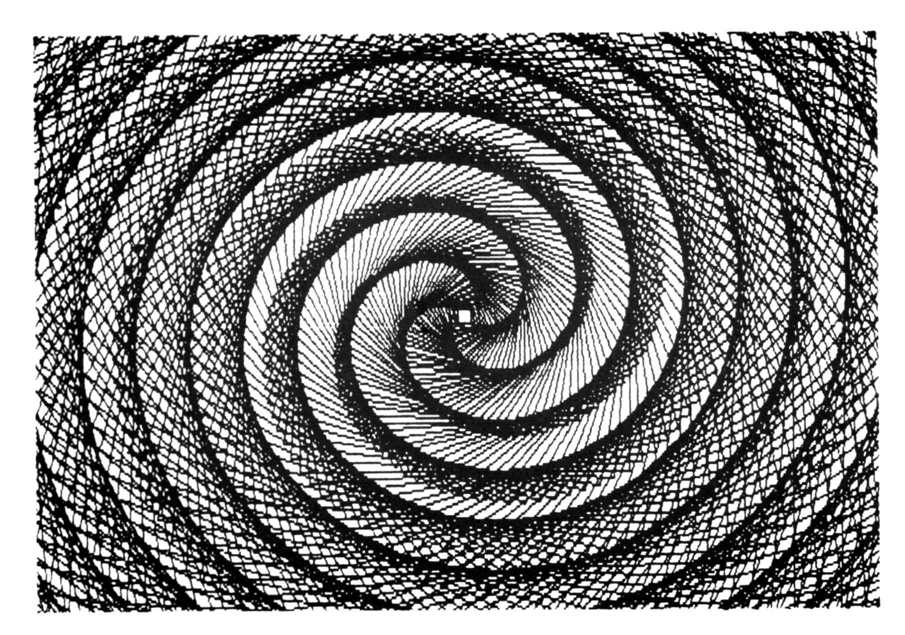


Figure 7.2: Résultat de POLYGONE.3 avec un angle de 89 degrés

La répétition avec récursion est par conséquent une solution élégante, même si elle n'est pas absolument indispensable parce que l'accroissement du pas cote se fait ici à travers l'appel de procédure. L'utilisation de make dans POLYGONE.3 est, au contraire, caractéristique de la programmation itérative de la même opération.

Les récursions employées jusqu'ici sont très simples: une procédure se termine en s'appelant elle-même. Ce type de récursion est facile à discerner et vous n'avez pas de raison d'hésiter à employer la récursion de cette façon. La programmation avec récursion est un outil puissant mais nous nous contenterons pour le moment d'une première introduction aux formes simples de récursion.

Malheureusement, la récursion n'est pas développée de façon très puissante dans le version du LOGO pour les ordinateurs AMSTRAD. Il existe pour des microordinateurs de même catégorie d'autres versions nettement supérieures sur ce plan!

7.6 Modèle de broderie: courbes désordonnées

A quoi pourrait-on donc parvenir avec un programme aussi simple que le suivant?

```
to BRODER :taille :n
fd :taille (rt :n)
BRODER :taille :n +1
end
```

Question banco: Que se passe-t-il lorsqu'on appelle BRODER 10 0? Laissez-vous surprendre!

L'opération de dessin se déroule de façon très simple: on exécute chaque fois un mouvement en avant de longueur fixe suivi d'une rotation sur la droite. L'angle augmente à chaque étape. Il s'ensuit une succession d'angles de rotation, cette succession étant produite par :n. Il s'agit simplement de la séquence des nombres naturels. On peut cependant utiliser au lieu de cela n'importe quelles autres séquences de nombres, ce qui pourra produire des images très variées.

Voici quelques exemples simples d'autres séquences de nombres:

```
a) :n * 10          b) :n * 10 +1
c) :n * 10 +3       d) :n * 10 +4
e) :n * 7           f) :n * :pi
```

Les exemples a) à f) doivent être ainsi compris: l'expression indiquée chaque fois doit remplacer, dans la seconde ligne de BRODER, l'appel de :n à la suite de rt. En faisant différents essais, vous constaterez que de petites modifications peuvent avoir des effets considérables sur les modèles de broderie produits. Les courbes sont cependant encore plus imprévisibles si on utilise des formules un peu plus compliquées:

```
g) :n * :n
```

Il n'y a pas ici de limites aux essais que vous pouvez faire. La taille des modèles obtenus peut être modifiée à votre guise avec la première entrée de BRODER. Vous pouvez utiliser en outre l'écran en boucle pour obtenir des effets encore plus curieux.

Si vous trouvez que le programme tourne trop lentement, ou si vous obtenez le message d'erreur I'm out of space (Je manque de place), vous pouvez aussi travailler avec une formulation itérative.

```
to BRODER :taille :nombre
local "n make "n 1
repeat :nombre [fd :taille rt :n * :pi make "n :n+1]
end
```



Figure 7.3: **Figures produites par BRODER**

Quelques mots en passant pour les lecteurs s'intéressant aux mathématiques: si on interprète le niveau du dessin comme niveau de nombres complexes, l'effet de BRODER peut être considéré comme la sommation de nombres complexes. La formule utilisée représente la phase du nombre complexe. Cette interprétation du niveau de dessin rend également très tentante une utilisation du graphisme avec tortue pour réaliser une méthode graphique pour ce qu'on appelle la transformation de Fourier.

7.7 Interruption sous condition

Dans des programmes de dessin, l'interruption manuelle, en appuyant sur des touches, peut être souhaitée expressément lorsqu'on n'a pas, par exemple, d'avance une idée précise de l'image devant être produite. Dans une répétition avec repeat, le nombre de ces répétitions est déjà fixé dès le départ. D'une manière générale, on souhaitera que l'exécution d'un programme soit interrompue

lorsqu'une condition déterminée sera remplie. Le cas le plus simple consiste à compter les répétitions. On obtient un effet semblable à celui de repeat:

```
to POLYGONE.2 :cote :angle
if :cote > 100 [stop]
fd :cote rt :angle ;ETAPE DE BASE
POLYGONE.2 :cote+1 :angle ;APPEL RECURSIF
end
```

La deuxième ligne est nouvelle. Elle est de la forme:

```
if ... [...]
si ... alors
```

if est suivi d'une comparaison, dans l'exemple ci-dessus:

```
:cote > 100.
```

Cette comparaison peut se terminer par la réponse vrai ou faux comme on peut le montrer en mode direct, en entrant une simple ligne d'entrée à titre d'exemple:

```
1 > 100
Réponse: FALSE (Résultat: faux)

99 = 99
Réponse: TRUE (Résultat: vrai)

4 < 10
Réponse: TRUE (Résultat: vrai)
```

Si le résultat de la comparaison après if est faux, alors le reste de la ligne de programme, jusqu'au prochain RETURN, sera ignoré. Si par contre la réponse est vraie, le reste de la ligne de programme sera traité normalement, y compris donc le contenu des crochets.

Dans notre exemple de programme c'est le mot LOGO

```
stop
```

entre crochets qui suit la comparaison. On comprend bien pourquoi puisque cela permet de mettre fin à l'exécution de la procédure. Si cette action a été elle-même appelée par une autre procédure, on passe alors à cette autre procédure. Le mot LOGO op (pour output = sortie) a un effet similaire. op met fin à la procédure mais avec sortie d'une valeur de résultat, contrairement à stop. On terminera une action avec stop ou avec op suivant la fonction de la procédure considérée. Une action se termine bien sûr aussi lorsque plus aucune instruction LOGO ne suit, comme par exemple dans les programmes graphiques élémentaires.

Polygones combinés

La simple répétition du déplacement en avant suivi d'une rotation peut déjà produire des figures très variées.

```
to COINS :cote :angle
COIN :cote :angle :angle
end

to COIN :cote :angle :total
fd :cote rt :angle
if (remainder :total 360 ) = 0 [stop]
COIN :cote :angle :total+:angle
end

to LUTINS :cote :angle
LUTIN :cote :angle :angle
end

to LUTIN :cote :angle :total
fd :cote rt :angle fd :cote rt 2*:angle
if (remainder :total 360 ) = 0 [stop]
LUTIN :cote :angle :total+:angle
end
```

Les véritables programmes de dessin sont ici constitués par COIN et LUTIN. COINS et LUTINS n'ont été formulés que pour faciliter les appels. COIN et LUTIN fonctionnent avec récursion.

La récursion est interrompue parce qu'on obtient toujours des figures fermées.

Faites donc des essais avec les angles les plus divers. Une modification de cote n'affecte que la taille mais pas la forme de la figure. Il faut toutefois rechercher chaque fois quelle longueur de côté est la plus intéressante pour un angle donné.

La modification de cote ou de angle donne des figures en spirale. La taille en entrée d détermine chaque fois la progression.

```
to SPIRALE :cote :angle :d
fd :cote rt :angle
SPIRALE :cote+:d :angle :d
end

to TOILE :cote :angle :d
fd :cote rt :angle
TOILE :cote :angle+:d :d
end
```

On obtient des figures plus complexes en combinant entre eux différents polygones. Nous choisirons ici une formulation itérative.

```
to ANGLESDOUBLES :s1 :w1 :s2 :w2 :n
local "w make "w :w2
repeat :n [rt :w1 fd :s1 rt :w fd :s2 lt :w make "w
:w+:w2]
end
```

s1, w1 et s2, w2 (w2+w1) sont respectivement les angles et les côtés de deux polygones différents. On exécute ici alternativement les étapes de base pour les deux polygones, le second angle étant produit par les deux rotations de w1 et w2.

Essayez un peu les combinaisons suivantes pour les valeurs en entrée:

```
(50 90 50 230 1),          (50 90 50 210 12),
```

(10 10 10 (-25) 72), (15 19 15 (-39) 360)

Vous devriez alors obtenir les dessins de la figure 7.4.

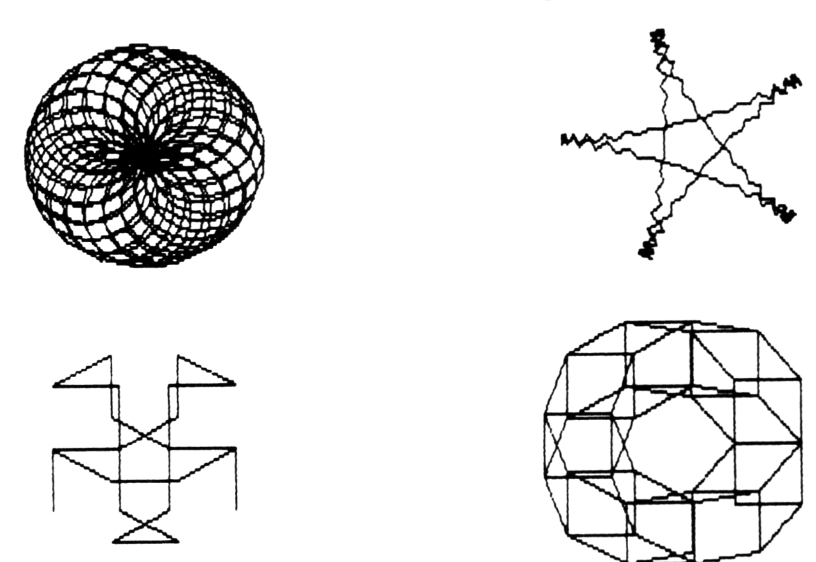


Figure 7.4: Dessins obtenus avec ANGLESDOUBLES

Exercices

1) Dessiner des cercles concentriques de rayons croissants.

2) Ecrire une procédure SOLEIL qui produira sur l'écran de nombreux rayons de soleil.

3) Le programme PHASESLUNAIRES devra représenter des croissants de lune correspondant aux différentes phases de la lune, de la pleine lune à la nouvelle lune.

4) Définir la procédure COURBEGAUCHE d'une façon similaire au programme COURBEDROITE.

Leçon 8: Mode d'emploi du système LOGO

Nouveaux éléments de vocabulaire:
save, load, dir, erasefile, changef, bye, savepic, loadpic, erasepic, dirpic, ct, clean, cs, recycle, nodes

8.1 Manipulation des disquettes

Les exemples de programme que nous avons vus jusqu'ici étaient tellement courts qu'il n'était pas très pénible de les taper au clavier. Au bout d'un certain temps de travail et d'expérimentation, vous aurez cependant certainement envie de pouvoir conserver les résultats de vos efforts en programmation, de façon à pouvoir les réutiliser ultérieurement. Il peut également arriver souvent que vous deviez interrompre une session de travail LOGO avant d'avoir atteint le but recherché. Dans ce dernier cas, il serait souhaitable de pouvoir disposer pour une session ultérieure de la situation qui prévalait au moment de l'interruption du travail.

Sauvegarde

Le vocabulaire de base de DR LOGO est sauvegardé sur une disquette système particulière qu'il est préférable de n'utiliser que pour cela. Le système LOGO actuel contient en outre, à un moment donné, les extensions de langage obtenues par programmation, c'est-à-dire les procédures que vous avez écrites vous-même ou que vous avez chargées à partir d'une disquette.

Le système LOGO comprend encore les noms globaux avec leurs valeurs respectives.

Pour 'sauver' la partie du système LOGO qui ne figure pas sur la

disquette système, on utilise la procédure LOGO save:

```
save "cercles
```

Ainsi sera stocké sur la disquette figurant actuellement dans le lecteur de disquette tout ce qui est actuellement disponible, aussi bien les extensions de langage que les noms. Le nom "cercles pourrait par exemple être utilisé pour sauvegarder les procédures de dessin de cercles que nous avons étudiées à la section précédente.

Sous DR LOGO, il n'est pas possible d'effacer avec save un fichier déjà existant sur la disquette. Si le nom utilisé avec cette instruction correspond à celui d'un fichier existant déjà sur la disquette, un message d'erreur sera sorti. Vous pouvez toutefois utiliser le programme SAUVEGARDER de la leçon 12. Ce programme change en effet le nom du programme déjà existant sur la disquette. Ce programme est donc mis en réserve. Le contenu actuel de la mémoire de travail peut alors être sauvegardé sous le nom demandé.

Si vous ne voulez sauvegarder qu'une partie des procédures ou noms créés au cours de la session LOGO, vous pouvez supprimer les parties indésirables. Vous trouverez également à la leçon 15 un programme utilitaire très pratique qui vous permettra de supprimer tous les programmes autres que ceux définis en entrée, ce qui est très avantageux à partir du moment où vous avez déjà accumulé de nombreuses procédures dont vous n'avez plus besoin (et que vous pouvez de toute façon avoir sauvegardées quelque part sur une disquette).

Avant de travailler avec save, il serait préférable d'examiner un peu ce qui est actuellement disponible:

```
poall
```

La sortie sera plus claire si vous ne faites sortir que les noms et les noms de procédures, avec:

```
pots pons
```

La suppression se fait avec le mot LOGO er (abréviation de erase = effacer, supprimer).

```
er "CERCLE.3
```

La procédure CERCLE.3 sera ainsi supprimée. Avec la ligne:

```
er [CERCLE.2 CERCLE.3 ROSETTE]
```

les procédures énumérées seront supprimées.

Avec:

```
ern "tauxtva
```

le nom global tauxtva sera supprimé du système. Pour supprimer plusieurs noms, il faut les réunir entre crochets.

On peut également supprimer d'un coup toutes les procédures ou tous les noms globaux, avec:

```
er glist ".def
```

ou

```
ern glist ".APV
```

Nous utilisons ici la fonction LOGO glist qui ne sera décrite en détail qu'au cours de la leçon suivante, en même temps que ce qu'on appelle les listes de propriétés.

Notez que, sous LOGO, tout le contenu du système à un moment donné peut être sauvegardé avec save. L'interruption momentanée d'une session LOGO ne pose donc aucun problème. Pour continuer, il suffit de recharger la dernière partie sauvegardée avec save. La situation de départ est alors rétablie.

save "cercles produit sur la disquette ce qu'on appelle un fichier LOGO. Si vous entrez un long nom de fichier, seuls les huit premiers caractères seront pris en compte. Le système ajoutera

automatiquement l'extension '.LOG' au nom de fichier, ce dont vous n'avez d'ailleurs pas à vous préoccuper à l'intérieur du LOGO. Ce n'est qu'au niveau du système d'exploitation CP/M que cette marque de fichier devient visible. Le fichier est pour le système d'exploitation un fichier de texte dont vous pouvez d'ailleurs consulter directement à l'écran ou faire sortir sur imprimante le contenu, même en dehors de DR LOGO (par exemple avec l'instruction CP/M type).

Le catalogue des fichiers LOGO disponibles

L'instruction

```
dir
```

réunit en une liste tous les noms des fichiers LOGO figurant sur une disquette. Les listes se reconnaissent au fait qu'elles sont placées entre crochets. Le catalogue de la disquette peut aussi être doté d'un nom et conservé sous ce nom.

```
make "catalog dir
```

Chargement à partir de la disquette

Un fichier LOGO peut être chargé avec la procédure load:

```
load "cercles
```

DR LOGO réagit à l'opération de chargement comme lors de la transmission de textes de programme après l'édition. Les différentes procédures sont maintenant annoncées comme ayant été définies. On peut charger successivement plusieurs fichiers LOGO, pour les incorporer au système LOGO, de même qu'on peut réaliser, avec l'éditeur, des textes de programmes en plusieurs étapes.

Lors du chargement on ne peut pas utiliser de caractères Joker. Le nom du fichier doit donc être entré littéralement et complètement pour chaque fichier à charger.

Suppression et modifications d'entrées sur la disquette

Le fichier LOGO cercles peut être supprimé avec:

```
erasefile "cercles
```

notez cependant que DR LOGO ne demande aucune confirmation lorsque vous employez erasefile. Il vaudrait donc mieux être prudent dans l'emploi de cette instruction.

Avant de supprimer définitivement un fichier, avec erasefile, on peut le conserver sous un autre nom, par mesure de précaution.

```
changef "premcerc "cercles
```

L'ancien fichier devrait ainsi recevoir le nom premcerc. Dans la version du LOGO dont l'auteur a pu disposer pour la rédaction de cet ouvrage, cette instruction ne fonctionne malheureusement pas. Le LOGO réagit en répondant:

```
File cercles not found
```

Il ne nous reste qu'à espérer que cette erreur du système soit rapidement éliminée par le fabricant.

Autres opérations avec les disquettes

Avant que vous puissiez y sauvegarder quoi que ce soit, les disquettes doivent d'abord avoir été formatées. Cela n'est possible qu'en dehors du LOGO, au niveau du système d'exploitation CP/M. Vous vous trouvez à ce niveau avant le chargement du système LOGO. Vous pouvez également revenir au système d'exploitation en terminant la session LOGO avec

bye

Notez cependant que le fabricant n'a, fort curieusement, pas jugé bon d'intégrer ici une demande de confirmation. Il n'y a donc pas de salut possible pour les programmes ou les données que vous

risqueriez de perdre si vous entriez cette instruction par erreur.

Les disquettes peuvent être formatées avec le programme utilitaire disckit ou disckit3. Vous devez utiliser pour sauvegarder les fichiers LOGO des disquettes de données (et non des disquettes système).

Le même programme utilitaire vous permet aussi de copier des disquettes. Il est recommandé de réaliser des copies de sécurité à intervalles réguliers.

Le fabricant a modifié le format de disquette; l'expérience révèle que les disquettes LOGO sont compatibles "vers le haut", c'est-à-dire que les disquettes dans l'ancien format peuvent être utilisés par les systèmes plus récents, par exemple sur le PCW pour la sauvegarde comme pour la lecture de fichiers LOGO. Les disquettes dans le nouveau format ne peuvent par contre pas être traitées par les systèmes plus anciens.

8.2 Chargement, sauvegarde et impression de dessins

Le contenu de la fenêtre graphique peut également être stocké directement sur la disquette et il peut ensuite être à nouveau chargé à partir de la disquette. Cela s'obtient respectivement avec les instructions savepic et loadpic.

```
savepic "modele cs
loadpic "modele
```

Lors du chargement d'une image, le LOGO passe de façon standard en mode d'écran divisé, c'est-à-dire que l'écran est divisé en une fenêtre graphique et une fenêtre de texte. Si vous voulez charger une image qui ait besoin de l'écran tout entier, il faut créer cet état avant le chargement.

```
fs cs ht loadpic "modele
```

Un dessin sur l'écran en haute résolution contient de nombreuses informations. Les fichiers correspondants nécessitent beaucoup de

place et il faut donc attendre plusieurs secondes aussi bien lors du chargement que de la sauvegarde.

Un dessin sauvegardé peut à nouveau être supprimé de la disquette avec l'instruction

```
erasepic "modele
```

Le catalogue des images stockées sur la disquette peut être obtenu avec l'instruction dirpic.

```
dirpic
```

Pour imprimer des dessins, les microordinateurs les plus récents disposent d'une fonction de hard copy intégrée qui permet de réaliser une copie conforme de l'écran sur l'imprimante. Sur le PCW AMSTRAD cette fonction est déclenchée par la combinaison de touches

```
EXTRA+PTR
```

C'est ainsi qu'ont par exemple été couchés sur le papier les dessins qui figurent dans le présent ouvrage.

Le CPC ne dispose malheureusement pas pour le moment de cette fonction de hard copy. Il ne reste donc plus que la possibilité de la photographie pour fixer sur le papier les images réalisées à l'écran. Il faut cependant faire un bon nombre d'essais avant de pouvoir sélectionner le bon éclairage. Une photo d'écran présente par ailleurs l'avantage de permettre aussi de restituer les couleurs si vous disposez d'un moniteur couleur.

Une autre possibilité consiste à faire sortir sur l'imprimante, au moyen d'un programme adapté, l'image stockée dans un fichier sur disquette.

8.3 Suppression

Vidage de l'écran

Lorsque vous écrivez des entrées dans la fenêtre de dialogue, que vous y recevez des réponses, vous parvenez bien vite au bord inférieur de l'écran. Si vous avez besoin de lignes supplémentaires, tout le contenu de l'écran est automatiquement décalé vers le haut par ce qu'on appelle le Scrolling de l'écran. La place nécessaire est ainsi libérée dans le bas de l'écran.

Si vous êtes gêné par tout ce qui est inscrit sur l'écran, vous pouvez effacer l'écran de texte avec:

ct (abréviation de clear text)

La fenêtre graphique peut être effacée avec:

clean

clean n'a pas d'effets autres. Si vous voulez que la tortue soit en même temps ramenée sur sa position de départ (y compris son orientation vers le 'Nord'), il vaut mieux utiliser plutôt:

cs (abréviation de clear screen)

Suppression de programmes

La place mémoire de votre ordinateur est limitée. Du fait de sa grande gamme de possibilités, le système LOGO lui-même occupe une grande quantité de place mémoire. D'un autre côté, le LOGO permet de formuler de façon très brève des programmes puissants. Il peut malgré tout vous arriver de manquer de place mémoire.

Lorsque vous éditez (c'est-à-dire consultez et modifiez) des procédures et que vous faites adopter de nouvelles versions, sous de nouveaux noms, dans le système LOGO, vous pouvez parvenir, au bout d'un certain temps, à une véritable accumulation de programmes

dont certains sont devenus totalement inutiles. Lorsque vous commencez à manquer de place, commencez par consulter, avec pots (abréviation de print out titles) la liste des procédures existantes. Vous pourrez alors recenser les noms de procédures inutiles.

Nous avons déjà expliqué au début de la leçon comment les procédures et les noms peuvent être supprimés avec les mots LOGO

```
er et ern
```

Les entrées sont chaque fois soit un nom de procédure ou de variable, qui doit alors être précédé de guillemets, soit une liste de tels noms placée entre crochets.

```
er "cercles , er [cercles impots], er glist ".DEF
ern "tauxtva , ern [pi a nombre], er glist ".APV
```

L'emploi de la fonction glist a également déjà été expliqué au début.

Elimination des déchets

Le système LOGO produit des déchets, c'est-à-dire de la place mémoire occupée de façon interne alors que ce n'est plus nécessaire. Cela ralentirait beaucoup le système s'il devait vérifier à chaque étape s'il ne pourrait pas libérer éventuellement telle ou telle place en mémoire. On peut cependant enclencher une 'élimination des déchets' avec l'instruction:

```
recycle
```

Avant de supprimer quoi que ce soit avec ERASE, il est donc conseillé d'essayer d'abord de libérer un peu de place par l'élimination des déchets. L'élimination des déchets que le système déclenche occasionnellement de lui-même peut provoquer des pauses gênantes. recycle permet par contre à l'utilisateur de décider lui-même du moment où doit se produire cette pause.

Place disponible

Vous pouvez vous informer à tout moment sur l'évolution de la place disponible pour le LOGO, avec:

```
nodes
```

nodes signifie noeuds. Un noeud est un peu l'unité de base pour les structures de données ou ici pour les programmes ou noms à traiter. Cette donnée ne vous sert véritablement que si vous l'examinez régulièrement. Si le nombre sorti dépasse 2000, il y a encore beaucoup de place disponible. 100 noeuds signalent évidemment que la place va bientôt commencer à manquer. Pour que cette donnée vous informe plus précisément, nous vous conseillons d'ordonner d'abord recycle.

8.4 Les fichiers de programmes avec chargement automatique

Les programmes utilisateur se composent souvent de plusieurs sections de programme. C'est pourquoi il est intéressant de faire exécuter automatiquement l'opération de chargement de tout un groupe de programmes liés entre eux. Sous DR LOGO, un fichier LOGO appelé STARTUP est recherché et chargé automatiquement, dès que le système LOGO est en place. Si vous voulez donc qu'une série de programmes soient chargés automatiquement, sans autre intervention, après le lancement du système LOGO, vous pouvez sauvegarder ces programmes (ou aussi le contenu de noms) sous le nom spécial de STARTUP. Si, lors du chargement du système LOGO à partir de la disquette système, un fichier STARTUP.LOG s'y trouve, il sera chargé sans autre instruction.

Cette option n'a d'intérêt que pour les programmes prêts à être utilisés. Cela permet par exemple d'écrire sur une disquette, à côté du vocabulaire de base LOGO, des extensions du langage de DR LOGO qui sont souvent utilisées. Il devient alors inutile de rechercher puis de charger tous ces programmes auxiliaires chaque fois qu'on charge et lance le système.

D'autres versions LOGO permettent en plus de faire lancer

automatiquement un programme sous le nom de STARTUP. Avec DR LOGO, une opération de lancement doit être appelée explicitement. Vous pouvez naturellement prévoir de toujours disposer, pour les paquets de programmes assez fournis, d'une procédure, qui aura toujours le même nom (STARTUP par exemple), permettant de lancer le programme.

8.5 Le travail avec deux lecteurs de disquette

Le lecteur de disquette que vous avez utilisé lors du chargement de DR LOGO est utilisé comme lecteur standard (Default disk) par les instructions load, save, erasefile. Si vous voulez appeler un second lecteur de disquette, vous devez le nommer explicitement.

```
dir "b:
```

Tous les fichiers LOGO de la disquette dans le lecteur de disquette B (second lecteur de disquette) seront sortis.

```
save "b:cercles
```

La mémoire de travail sera sauvée sous le nom cercles.log sur le lecteur de disquette B.

```
load "b:impots
```

Le fichier LOGO impots sera chargé à partir du lecteur de disquette B.

Il est également possible, avec:

```
setd "b:
```

de diriger toutes les opérations sur le lecteur de disquette indiqué. On évite ainsi d'avoir à fixer chaque fois le nom du lecteur de disquette devant le nom de fichier.

Le PCW AMSTRAD dispose également de ce qu'on appelle un disque RAM qui porte le nom m. Il s'agit en fait d'une zone mémoire

dans l'ordinateur à laquelle on peut accéder de la même façon que l'on accède à un lecteur de disquette. Un lecteur de disquette est donc simulé mais avec une vitesse d'accès bien sûr très supérieure à ce qu'elle est sur un véritable lecteur de disquette. Comme les programmes LOGO prennent peu de place, cet avantage n'est toutefois pas décisif sous LOGO.

Leçon 9: Mots et listes

Nouveaux éléments de vocabulaire:
item, count, piece, se, show, first, bf, last, bl, emptyp, fput, lput, memberp, where, numberp, wordp, listp, equalp, not, shuffle

Programmes:
DECOMPOSER, BAVARDER, COIN, RENVERSER, RENVERSER.PHRASE, DADA, BEGAYER, ECHANGE.LISTES, LOTO, TIRAGE, BOULES, REMPLACE

9.1 Les chaînes de caractères comme mots

Dans une langue, des phrases sont formées à partir de mots, les mots étant eux-mêmes composés de lettres. Pour former d'autres groupes significatifs, les lettres de l'alphabet ne suffisent pas. C'est pourquoi nous utilisons en renfort des caractères spéciaux tels que la virgule, le point, le point-virgule, les guillemets, etc... Sur le clavier de votre ordinateur, vous trouvez également, à côté des lettres, un nombre assez important de caractères spéciaux, y compris bien sûr les dix chiffres. Pour l'ordinateur, certains caractères ont une signification particulière mais tous les caractères sont traités, au départ, de la même façon.

Pour le LOGO, n'importe quelle suite de caractères, lettres, chiffres et caractères spéciaux constitue un mot.

Exemples:

```
"SOLEIL
"ORDINATEUR
"3.14159
"<<..>>
```

Seuls les deux premiers mots pourraient également être des mots du langage courant. Dans les langages informatiques, ce qui importe le

plus, dans un premier temps, ce n'est pas la signification intrinsèque d'une expression mais sa conformité aux règles du langage.

Tous les mots commencent par des guillemets. Nous avons déjà expliqué, tout au début, le rôle que jouent les guillemets pour les noms. Les guillemets ne font pas partie du mot lorsqu'ils sont placés tout au début. Ils indiquent simplement que les caractères suivants constituent un mot. Il est facile de vérifier, avec PRINT, ce que LOGO a accepté comme mot:

```
pr "SOLEIL
pr "HOURRA"
pr "DEUX MOTS
```

Dans le premier exemple, SOLEIL devrait apparaître sur l'écran. "HOURRA" dans le second. Pour le troisième exemple, vous obtenez un message d'erreur:

```
I don't know how to MOTS
```

DEUX a été accepté comme mot. MOTS est par contre interprété comme un nom de procédure et le message d'erreur indique qu'il n'existe pas d'action sous ce nom.

> Le début d'un mot est indiqué au moyen des guillemets qui ne font pas partie du mot lui-même. Tous les caractères suivants jusqu'à la prochaine marque de séparation forment le mot.

C'est le caractère espace qui fait office de marque de séparation sous LOGO. A la fin d'une ligne, il est superflu de placer une marque de séparation.

```
(pr "DEUX "MOTS)
```

C'est ainsi qu'on peut faire sortir DEUX MOTS. Les parenthèses sont nécessaires parce que pr n'attend normalement qu'une seule entrée. Notez bien que la parenthèse refermée constitue également une marque de séparation et qu'elle ne fait pas partie du mot. DR LOGO

exécutera cette ligne correctement même si vous omettez la seconde parenthèse. Les parenthèses sont automatiquement refermées à la fin d'une ligne, de même que les crochets d'ailleurs.

Comme dans une langue naturelle, l'espace ne peut constituer un caractère à l'intérieur d'un mot. Il est toutefois possible d'inhiber l'effet de séparation du caractère espace. Il faut pour cela que l'espace soit précédé d'un caractère backslash, c'est-à-dire du trait de fraction renversé:

```
pr "DEUX\ MOTS
```

Ce caractère est indiqué sur le clavier du CPC; il peut cependant également être produit par la combinaison de touches CONTROL+Q ou ALT+Q sur le PCW.

9.2 Les phrases sous forme de listes

A partir de plusieurs mots, on peut constituer une phrase complète, les mots étant habituellement séparés entre eux, dans une phrase, par des espaces.

Une suite de mots séparés par des espaces constitue pour le LOGO un exemple simple de liste. Les listes sont placées entre crochets, les crochets ne faisant pas eux-mêmes partie de la liste mais en indiquant simplement le début et la fin.

```
[NOUS SOMMES DIMANCHE]
[10 25 27 31 32 46]
```

La première liste se compose de trois mots, la seconde de six. Une liste peut être sortie en bloc:

```
pr [NOUS SOMMES DIMANCHE]
```

La réponse de DR LOGO sera, comme prévu:

```
NOUS SOMMES DIMANCHE
```

Les crochets n'ont pas été sortis. Nous obtiendrions la même sortie avec:

```
(PRINT "NOUS "SOMMES "DIMANCHE )
```

La ligne avec la liste constitue visiblement la variante la plus pratique.

On ne peut pas voir d'après la sortie obtenue si plusieurs mots ont été sortis les uns après les autres ou si c'est une liste composée de plusieurs mots qui a été sortie d'un seul coup. Si vous voulez pouvoir voir sur l'écran si c'est une liste qui a été sortie, vous avez la possibilité d'utiliser la procédure LOGO show au lieu de pr:

```
show [NOUS SOMMES DIMANCHE]
```

Les crochets caractéristiques des listes seront alors sortis avec le texte.

Les listes ne servent bien sûr pas uniquement à être sorties sur l'écran ou imprimées. Il existe au contraire toute une série de mots LOGO permettant de manipuler des listes.

Les listes peuvent par exemple être attribuées à un nom. Ce nom lui-même est un mot.

```
make "LISTE "[NOUS SOMMES DIMANCHE]
```

La liste [NOUS SOMMES DIMANCHE] sera ensuite disponible sous le nom de LISTE.

```
pr :LISTE
```

Remarquez le double point devant LISTE; il indique qu'il s'agit du contenu du tiroir appelé LISTE. Cela veut donc dire: imprimer la valeur de LISTE!

Vous pouvez d'ailleurs également doter de simples mots d'un nom permettant de les appeler:

```
make"MOT"SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS
```

Cela vous permettra de n'utiliser qu'une abréviation dans vos programmes, tout en pouvant ainsi faire sortir le mot en entier.

La première fois que nous avons rencontré les listes, c'était par rapport à la répétition avec repeat:

```
repeat 10 [pr 1+random 6]
```

Une liste peut donc aussi contenir des noms de procédures. La liste repeat indiquée peut également être dotée d'un nom pour être ultérieurement utilisée sous ce nom:

```
make "LISTE.REP [pr 1+random 6]
repeat 10 :LISTE.REP
```

9.3 Phrases de phrases: les listes hiérarchisées

On peut former des phrases avec des mots. Plusieurs phrases peuvent à leur tour former tout un chapitre, plusieurs chapitres un livre, plusieurs livres une bibliothèque.

On établit ainsi une hiérarchie dont l'unité de base est constituée par le mot. Dans le sens inverse, on pourra ensuite rechercher dans une bibliothèque le 500ème livre, dans ce livre le 7ème chapitre, dans ce chapitre la troisième phrase, dans la phrase elle-même le second mot. On peut ainsi accéder indépendamment à chaque unité (livre, chapitre, phrase, mot) comme si elle constituait un tout.

La structure obtenue de cette façon, à partir des mots, est appelée une liste. Les phrases que nous avons vues jusqu'à présent, dans des listes pour une sortie de texte avec pr ou avec la liste repeat, constituent le niveau immédiatement au-dessus des mots eux-mêmes. La possibilité de construire une hiérarchie de niveaux fait des listes un instrument extrêmement puissant.

Nous nous intéresserons plus tard aux structures de listes complexes. Nous allons d'abord expliquer les principales opérations

avec les listes et les mots au moyen d'exemples simples.

9.4 Décomposition de listes et de mots

Les listes et les mots peuvent être manipulés à volonté dans un programme. Il faut pour cela des opérations de base de décomposition et de fusion qui puissent être appliquées aussi bien aux listes qu'aux mots.

Les mots et les listes peuvent tout d'abord être décomposés en leurs différents éléments de même qu'une phrase peut être décomposée en différents mots et un mot en différentes lettres. Pour isoler un élément déterminé, on utilise la fonction ITEM:

```
make "PHRASE [LE SOLEIL BRILLE]
item 2 :PHRASE
```

Réponse:
SOLEIL

L'opération item fournit également un caractère déterminé d'un mot:

```
make"MOT"SOCIETENATIONALEDESCHEMINSDEFERFRANCAIS
item 25 :MOT
```

Réponse: N

Il peut arriver que l'élément demandé n'existe pas parce que la liste ou le mot sont trop petits.

```
item 5 :PHRASE
```

Vous obtenez le message d'erreur:

```
Too few items in [LE SOLEIL BRILLE]
```

(Il y a trop peu d'éléments dans la liste ...)

Le nombre d'éléments peut être déterminé à l'aide de la fonction LOGO count.

```
count :PHRASE
Réponse: 3

count :MOT
Réponse: 36
```

Décomposition en avant et en arrière

Nous pouvons écrire dès maintenant une instruction qui décompose une liste ou un mot en tous ses éléments et sorte ceux-ci les uns sous les autres.

```
to DECOMPOSER :objet :n
if :n > count :objet [stop]
(pr :n item :n :objet)
DECOMPOSER :objet :n+1
end
```

La procédure DECOMPOSER s'appelle elle-même, par récursion, pour déterminer le prochain élément. Dans la seconde ligne, l'opération est interrompue si le nombre d'éléments est déjà dépassé par la variable de comptage n.

```
DECOMPOSER :PHRASE 1
```

Le LOGO vous répondra:

```
1 LE
2 SOLEIL
3 BRILLE
```

Dans la troisième ligne de DECOMPOSER, la variable de comptage n est sortie avec l'élément correspondant, pour plus de clarté. Pour le long mot SOCIETE..., l'appel commencera seulement à partir du 31ème élément pour ne pas trop prendre de place.

```
DECOMPOSER :MOT 32
```

Réponse:

```
32 F
33 R
34 A
35 N
36 C
37 A
38 I
39 S
```

Le programme suivant écrit sur l'écran une liste ou un mot dans l'ordre inverse des éléments:

```
to ENARRIERE :objet :n
if n = 0 [stop]
type item :n :objet
ENARRIERE :objet :n-1
end
```

Si vous appliquez cette procédure à des mots, avec:

```
ENARRIERE :MOT count :MOT
```

vous obtenez sur l'écran le mot renversé. La sortie est faite avec type pour que les différents éléments soient accolés les uns aux autres.

Examinez également l'entrée. L'objet est décomposé en sens inverse, en commençant par l'élément dont le numéro est donné par la seconde valeur en entrée. C'est pourquoi nous avons transmis ici, lors de l'appel, le résultat de la fonction count comme seconde valeur en entrée.

Si une liste, par exemple PHRASE doit être écrite en sens inverse, il est préférable de faire imprimer des espaces entre les différents éléments. A cet effet, la troisième ligne de ENARRIERE peut être ainsi modifiée:

```
(type item :n :objet "\ )
```

Après l'appel ENARRIERE :PHRASE count :PHRASE, vous

devriez alors obtenir à l'écran:

```
BRILLE SOLEIL LE
```

Sections de mots et de listes

item isole un élément, et un seul, d'un mot ou d'une liste. Il arrive cependant souvent qu'on souhaite extraire toute une section d'un mot ou d'une liste. Le LOGO met à votre disposition un mot spécial à cet effet:

```
piece 8 16 :MOT
Réponse: NATIONALE
```

piece sort une section de la liste ou du mot indiqués, les deux premières entrées de piece indiquant le début et la fin de la section.

Les nombres sont aussi des mots pour DR LOGO. Essayez par exemple:

```
make "pi 3.1415926535898 piece 3 7 :pi
Réponse: 14159
```

On sort ici les chiffres 3 à 7 du nombre pi. La comparaison avec le résultat de pi montre que le point décimal est également pris en compte. Nous avons en effet obtenu les caractères 3 à 7 du mot 3.141592.

Faites maintenant un essai avec une liste en entrée:

```
piece 2 3 :PHRASE
Réponse: [SOLEIL BRILLE]
```

Les crochets indiquent que le résultat est une liste. Avec une liste hiérarchisée du niveau au-dessus, c'est-à-dire avec un paragraphe composé de plusieurs phrases:

```
make "PARAGRAPHE
[[NOUS SOMMES DIMANCHE] [LE TEMPS EST DETESTABLE]
```

[JE NE VEUX PAS] [NOUS ATTENDONS GODOT]]

piece 2 3 :PARAGRAPHE

piece produira comme résultat un paragraphe formé des deuxième et troisième phrases:

[[LE TEMPS EST DETESTABLE] [JE NE VEUX PAS]]

Notez la différence entre le résultat de item et celui de piece:

- Avec piece, le résultat est de même nature que la troisième entrée; une section de mot donne un mot, une section de liste une liste.

- Le résultat de item est par contre un élément subordonné car on descend alors d'un niveau dans la hiérarchie. Pour un mot, on obtiendra donc un caractère, pour un paragraphe une phrase.

9.5 Fusion de mots et de listes

Les phrases

La procédure ENARRIERE décompose une phrase à partir de la fin et construit ainsi une nouvelle phrase sur l'écran. Mais on ne peut rien faire du résultat obtenu car il n'a qu'une existence provisoire, en tant qu'image d'écran.

Pour construire une nouvelle phrase, c'est-à-dire une liste composée de mots, on utilise le mot LOGO

se (abréviation de sentence = phrase)

se "BONNE "JOURNEE

On produit ainsi une liste composée de deux mots, ici la phrase BONNE JOURNEE. C'est pourquoi le LOGO vous répond:

```
[BONNE JOURNEE]
```

Les crochets indiquent que le résultat est une liste.

se attend normalement deux entrées. Si une phrase doit se composer de plus de deux mots, le nom de procédure doit être placé, avec les entrées, entre parenthèses:

```
(se "LE "SOLEIL "BRILLE)
```

Réponse: [LE SOLEIL BRILLE]

Formule d'adresse

```
to MONSIEUR :nom
op se [Cher Monsieur] :nom
end
to MADAME :nom
op se [Chere Madame] :nom
end
```

Appel:

```
pr MONSIEUR "LOGO
```

Réponse:

```
Cher Monsieur LOGO
```

Pour bien comprendre ce qui se passe, voici un autre appel avec show au lieu de pr:

```
show MADAME "Durand
```

Réponse:

```
[Chere Madame Durand]
```

show sort aussi les crochets, le résultat de MADAME, comme d'ailleurs celui de MONSIEUR, est donc une liste. Il s'agit cependant d'une forme spéciale de liste, à savoir d'une phrase

comme on pouvait d'ailleurs s'y attendre puisque se veut dire phrase. La première entrée est elle-même une phrase, la seconde est un mot.

Entrez maintenant:

```
show MONSIEUR [LOGO]
```

Ici aussi, c'est une phrase qui est sortie bien que la seconde entrée ne soit pas un mot mais une liste (à cause des crochets).

Lorsqu'une entrée de se n'est pas un mot mais une liste, le LOGO élimine alors les crochets et seulement les crochets.

Les mots

La procédure LOGO word permet de former des mots comme on forme des phrases avec se:

```
show word "ALLE "LUIA
```

Le LOGO vous répond ALLELUIA. Vous voyez bien qu'il s'agit effectivement d'un mot et non d'une liste puisque même show ne sort pas de crochets. Comme on obtient ainsi un mot unique, il n'y a qu'un élément et par conséquent il n'y a pas d'espace de séparation.

Comme pour se, il faut utiliser des parenthèses pour réunir en un mot plus de deux entrées.

```
(word "CHEMINS "DE "FER)
```

S'ajoutant à la décomposition de mots, la fusion avec word permet toutes sortes de manipulations de mots.

On peut ainsi aboutir à des applications dans le domaine du langage:

je marche, tu marches, il marche, nous marchons, vous marchez, ils

marchent

Tout cela peut être créé à partir de la forme de base 'marcher'. Bien entendu, dans les langues naturelles, les verbes irréguliers rendent très difficile la programmation de tous les verbes.

```
to PLURIEL :singulier
op word :singulier "s
end
```

Appel:

```
PRINT PLURIEL "rue
```

Réponse:

```
rues
```

La formation du pluriel est malheureusement loin d'être toujours aussi simple et le programme PLURIEL ne maîtrise qu'un type de formation du pluriel même si c'est de loin le type le plus répandu!

Comme les nombres aussi sont des mots pour LOGO, word permet aussi de constituer des nombres.

```
(word 3 ".14159)
```

Réponse: `3.14159`

La manipulation de mots et de nombres sous LOGO offre plus de possibilités qu'on ne pourrait le croire à première vue. Les mots LOGO eux-mêmes, aussi bien les mots du vocabulaire de base que ceux qui ont été définis en plus par la programmation de procédures, sont des mots, le texte du programme des procédures est une liste. On a ainsi des possibilités extrêmement vastes de manipulation des programmes par des programmes. Nous y reviendrons à la leçon 17.

La combinaison de word ou se avec piece permet de fusionner plusieurs sections de mots ou de listes:

```
word piece 1 7 :MOT piece 32 39
```

Réponse: SOCIETEFRANCAIS

ou:

```
(se piece 1 2 :PHRASE [TOUT LE] piece 2 3 :PHRASE)
```

Réponse: [LE SOLEIL TOUT LE SOLEIL BRILLE]

9.6 Phrases aléatoires

Avec les nombres aléatoires, beaucoup de programmes simples peuvent produire des effets très amusants. Un programme produisant des phrases en français dont le contenu serait aléatoire tout en respectant les règles de grammaire, voilà qui deviendrait vite très compliqué à réaliser si vous autorisiez une grande variété de constructions de phrases possibles.

Nous nous limiterons donc à une structure de phrases élémentaire, du type:

```
Antoine rentre vite chez lui
```

Cela devient déjà plus compliqué si le sujet peut apparaître au pluriel ou si on veut utiliser d'autres temps que le présent, etc...

```
to BAVARDER
pr (se SUJET VERBE ADVERBE COMPLEMENT.LIEU)
end
```

C'est un programme-cadre qui fixe la structure de la phrase: sujet, verbe, adverbe et complément de lieu.

La phrase sera produite par les sorties des procédures SUJET, VERBE, ADVERBE, COMPLEMENT.LIEU.

```
to SUJET
op "ANTOINE
end
to VERBE
op "RENTRE
end
to ADVERBE
op "VITE
end
to COMPLEMENT.LIEU
op [CHEZ LUI]
end
```

Il ne s'agit évidemment pas de procédures très sophistiquées puisqu'elles se contentent de sortir un mot ou une liste comme résultat.

Même des procédures aussi simples peuvent être intéressantes. Elles peuvent en effet vous éviter de devoir affecter des valeurs à des noms globaux avec make. Pour les procédures, on n'écrit que le nom de la procédure, alors qu'avec un nom SUJET, auquel on affecterait par exemple le nom ANTOINE, avec:

```
make "sujet "ANTOINE
```

il faudrait utiliser la valeur de sujet et donc :sujet.

La sortie de COMPLEMENT.LIEU n'est pas un mot mais une liste ayant la forme simple d'une phrase. Lors de l'appel de BAVARDER, se supprime les crochets qui entourent la liste. Le sujet pourrait de même se composer de plusieurs mots:

```
to SUJET
op [LE PAUVRE ANTOINE]
end
```

Après que vous ayez entré BAVARDER, vous devriez maintenant voir sur l'écran:

```
LE PAUVRE ANTOINE RENTRE VITE CHEZ LUI
```

Il existe une autre façon de regrouper plusieurs mots. Au lieu de constituer une phrase (au sens du LOGO), vous pouvez réunir plusieurs mots en un seul en spécifiant que les espaces séparant les mots devront être traités comme des caractères normaux et non comme des marques de séparation:

```
"LE\ PAUVRE\ ANTOINE  au lieu de [LE PAUVRE ANTOINE]
```

Pour le moment, il n'y a pas beaucoup de bavardage et nous nous contentons de sortir toujours la même phrase. Il faudrait donc que les procédures pour les différentes parties de la phrase possèdent chacune une provision de réponses possibles dans laquelle elles piocheraient au hasard.

```
to SUJET
op item 1+random 10 [JEAN [LE PAUVRE ANTOINE] [LE
GOUVERNEMENT][UNELEPHANT][LECAMION]ILTONTONPAPA[LE
CHIEN] PEPE]
end
```

Bavarder devra bien sûr produire un grand nombre de phrases:

```
to BAVARDER
pr (se SUJET VERBE ADVERBE COMPLEMENT.LIEU)
BAVARDER
end
```

Vous pourrez maintenant obtenir,par exemple, à l'écran:

```
LE GOUVERNEMENT RENTRE VITE CHEZ LUI
UN ELEPHANT RENTRE VITE CHEZ LUI
IL RENTRE VITE CHEZ LUI
```

De la même manière que pour le sujet, les procédures pour les autres parties de la phrase devront être dotées d'une série de possibilités pour qu'on obtienne réellement un effet de surprise.

```
to VERBE
op item 1+random 8 [COURT [SE PRECIPITE] ARRIVE PARVIENT
REVIENT ROULE TOMBE APPARAIT]
```

```
end
```

Nous laissons maintenant le champ libre à votre imagination pour ce qui est d'achever ce programme.

9.7 Découpage de listes et de mots

item vous permet d'isoler des éléments isolés d'une liste ou d'un mot en indiquant le numéro de l'élément voulu. piece vous permet d'extraire des parties entières d'un mot ou d'une liste.

Un autre type de décomposition consiste à couper un élément au début ou à la fin d'une liste ou d'un mot.

Pour en faire l'essai, nous allons définir une liste test avec une procédure:

```
to ESSAI
op [le logo est un langage super]
end
```

Entrez maintenant:

```
pr first "auto
pr first essai
pr last "bicyclette
pr last essai

pr bf "auto
pr bf essai
pr bl "bicyclette
pr bl essai
```

Réponses:

```
a
le
e
super
```

```
uto
logo est un langage super
bicyclett
le logo est un langage
```

Vous pouvez aisément déduire de ces exemples l'effet des mots LOGO que nous venons de vous présenter.

- first sort le premier élément d'une liste ou le premier caractère d'un mot,
- last le dernier élément ou caractère.
- (bf abréviation de but first) sort tout sauf le premier élément d'une liste ou le premier caractère d'un mot,
- bl (abréviation de but last) sort tout sauf le dernier élément ou caractère.
- Les mots LOGO first, last, bf, bl attendent une entrée qui doit être une liste ou un mot.

Ces instructions aussi sont plus puissantes qu'on ne pourrait le supposer à première vue. Si vous essayez par exemple:

```
pr last bl ESSAI
pr first last bl ESSAI
pr bl bl bl ESSAI
```

L'effet du découpage se développe donc considérablement si on enchaîne les découpages. Lorsqu'on supprime progressivement des parties d'une liste, au début ou la fin, on peut composer les combinaisons les plus diverses.

L'enchaînement nous donnera dans le premier exemple:

> bl ôte d'abord le dernier élément de la liste formée par la procédure ESSAI; il reste [LE LOGO EST UN LANGAGE]. last extrait maintenant le dernier élément du reste et sort ainsi l'avant-dernier mot: LANGAGE.

L'enchaînement des opérations peut notamment être réalisé par l'appel récursif de procédures:

```
to COIN :objet
pr :objet
COIN bf :objet
end
```

L'appel avec COIN "CARABOSSE nous donnerait:

```
CARABOSSE
ARABOSSE
RABOSSE
ABOSSE
BOSSE
OSSE
SSE
SE
E
```

et le message d'erreur:

```
bf doesn't like an empty word as input in:
COIN bf :objet
```

Le message d'erreur indique que bf a rencontré un objet vide qu'il n'est donc plus possible de décomposer.

Qu'obtiendrons-nous avec:

```
bf "E
```

Si le premier élément est ôté, il ne reste plus rien. Le LOGO acceptera malgré tout cette instruction mais la réponse sera une ligne vide puisque le résultat n'a pas de contenu!

Le nombre d'éléments du résultat est d'ailleurs effectivement zéro comme nous pouvons le vérifier avec:

```
count bf "E
```

Réponse: 0

On peut donc dire qu'on a affaire à ce qu'on appelle un ensemble vide, c'est-à-dire un ensemble contenant zéro élément. Il existe donc bien des mots vides ou des listes vides:

```
to MOT.VIDE
op "
end
to LISTE.VIDE
op []
end
```

MOT.VIDE produit le même résultat que bf "E. Le mot vide et la liste vide n'ont de contenu ni l'un ni l'autre mais ils se distinguent quand même par leur type.

Pour éviter un message d'erreur dans la procédure COIN, il faut prévoir une interruption programmée de la récursion. On peut insérer à cet effet une ligne if dans le programme:

```
to COIN :objet
if :objet = " [stop]
pr :objet
COIN bf :objet
end
```

Ce test ne peut cependant être utilisé que si OBJET est un mot car une liste vide n'est pas la même chose qu'un mot vide.

```
COIN ESSAI
```

donnera également le message d'erreur:

```
bf doesn't like [] as input in ...
```

La procédure n'a pas été arrêtée par stop alors que le contenu de objet était bien vide. Pour que la procédure puisse s'appliquer à des listes, l'interrogation IF devra se présenter ainsi:

```
if :objet = [] [stop]
```

Mais il existe heureusement une solution qui fonctionnera dans les deux cas:

```
to COIN :objet
if emptyp :objet [stop] pr :objet
COIN bf :objet
end
```

Ici apparaît pour la première fois un mot LOGO qui débouche sur un point d'interrogation. Ces mots sont des mots de test qui permettent de tester si une condition est remplie.

emptyp :objet posera la question:

Le contenu de objet est-il vide?

La réponse ne pourra être que VRAI ou FAUX.

```
emptyp "CARABOSSE
```

Réponse: FALSE (Résultat: faux)

De même pour emptyp ESSAI.

Aux entrées suivantes:

```
emptyp "
emptyp []
```

le LOGO répondra par contre:

TRUE (Résultat: vrai)

La condition formulée avec le mot de test emptyp permet d'interrompre la procédure quel que soit le type de l'entrée, mot ou liste.

```
COIN ESSAI
```

Réponse:

```
le logo est un langage super
logo est un langage super
est un langage super
un langage super
langage super
super
```

Dans nos essais avec les mots LOGO first, last, bf, bl, un petit détail nous a échappé du fait que la sortie avec pr supprime automatiquement les crochets extérieurs.

```
show first ESSAI
show butfirst ESSAI
```

Réponse:

```
le
[logo est un langage super]
```

Avec une phrase telle que celle produite ici par ESSAI, first tire de la liste un mot comme résultat alors que le résultat de bf est à nouveau une liste comme l'indiquent les crochets. Dans la hiérarchie mot, phrase, chapitre, livre, bibliothèque, first et last descendent au niveau immédiatement inférieur alors que bf et bl restent au même niveau.

```
first first [UNE PHRASE]
bf bf [UNE PHRASE]
```

Dans le premier exemple nous obtenons la lettre U comme résultat alors que, dans le second exemple, c'est une liste vide [] qui sera produite.

Renversement de mots

Pour renverser un mot sur l'écran, il suffit d'en imprimer les différentes lettres en commençant par la fin. Les lettres de devant

doivent être envoyées à la fin et les lettres finales doivent être envoyés devant pour que le mot renversé soit à notre disposition.

```
to A.LA.FIN :MOT
op WORD (bf :MOT) (first :MOT)
end
```

Appel: pr A.LA.FIN "SALUT

Réponse: ALUTS

Cette opération renverserait effectivement le mot si le reste produit avec bf était déjà renversé auparavant. La récursion peut maintenant nous permettre de parvenir à une solution élégante.

Avant que le reste du mot soit à nouveau fusionné à la lettre envoyée en arrière, ce même processus doit être appliqué au reste lui-même.

Il faut donc remplacer:

bf :MOT par A.LA.FIN bf :MOT

La répétition provoquée par l'auto-appel doit être interrompue de façon contrôlée. Si une ligne ne se charge pas de l'interruption, un message d'erreur apparaîtra lorsque le mot plusieurs fois décomposé ne pourra plus l'être, c'est-à-dire lorsqu'un mot vide aura été produit.

```
to RENVERSER :MOT
if emptyp :MOT [op :MOT]
op word (RENVERSER bf :MOT) (first :MOT)
end
```

Appel: RENVERSER "SALUT

Réponse: TULAS

Le transfert de la lettre ne commence maintenant que lorsque la décomposition atteint la dernière lettre du mot entré, O en

l'occurrence.

Renversement de phrases

Pour renverser des phrases selon le même principe, il suffit de remplacer word par se dans le programme de renversement de mots.

```
to RENVERSER.PHRASE :PHRASE
if emptyp :PHRASE [op :PHRASE]
op se (RENVERSER.PHRASE bf :MOT) (first :PHRASE)
end
```

Appel: pr RENVERSER.PHRASE [COMMENT CA VA]

Réponse: VA CA COMMENT

Appel: pr RENVERSER.PHRASE ESSAI

Réponse: super langage un est logo le

Pour soigner la présentation, nous avons également remplacé MOT par PHRASE mais cela n'a aucun effet sur le programme.

9.8 Rallonger des listes

A first et last correspondent des fonctions inverses, fput et lput (fput et lput signifient first put et last put).

Exemples:

```
show fput "BONNE [JOURNEE]
show fput "QUELLE [HEURE EST IL ?]

show lput "CHEZ [SOI ON EST BIEN]
show lput "QUELLE? [HEURE EST IL]
```

Réponse:

```
[BONNE JOURNEE]
[QUELLE HEURE EST IL ?]

[SOI ON EST BIEN CHEZ]
[HEURE EST IL QUELLE?]
```

La sortie avec show montre que fput et lput produisent systématiquement des listes. Ces deux instructions ne sont donc pas tout à fait symétriques à first et last qui peuvent en effet aussi s'appliquer à des mots.

fput et lput attendent deux entrées. La seconde doit être une liste alors que la première peut être une liste ou un mot. La tâche de ces instructions consiste donc à ajouter un mot ou une liste entière au début ou à la fin d'une liste existante.

Dans certains cas, le même effet peut être obtenu avec d'autres mots LOGO:

```
show se "BONNE "JOURNEE
show se "BONNE [JOURNEE]
show se [BONNE] [JOURNEE]
```

La réponse sera chaque fois: [BONNE JOURNEE]

Le résultat est donc chaque fois identique à celui de fput "BONNE [JOURNEE]. Le second exemple est entièrement conforme à l'emploi de fput. L'entrée du premier exemple ne serait pas valable pour fput parce que la seconde entrée doit être une liste.

```
show se [BONNE] [JOURNEE]
show fput [BONNE] [JOURNEE]
```

Réponses: [BONNE JOURNEE] dans le premier cas et [[BONNE] JOURNEE] dans le second.

Ici apparaît une autre différence par rapport à se. fput, contrairement à se, ne supprime pas les crochets de la première entrée. C'est donc non pas un mot mais une liste qui formera le premier élément.

lput fonctionne exactement comme fput si ce n'est que la première entrée est ajoutée à la fin de la liste attendue comme seconde entrée.

Applications de fput et lput

fput a été conçu comme le contraire de first.

```
to ESSAI
op [LE LOGO EST UN LANGAGE SUPER]
end

pr fput (first ESSAI) (bf ESSAI)
```

Réponse: le logo est un langage super

first et bf décomposent la liste produite par ESSAI en son premier élément, 'le' en l'occurrence et en un reste. fput réunit à nouveau ces deux parties ce qui donne la liste de départ de ESSAI.

Cela devient très intéressant lorsqu'on commence à combiner les opérations qui agissent sur le début d'une liste avec celles qui agissent sur la fin d'une liste. C'est-à-dire fput avec last, lput avec first et des combinaisons comprenant toutes ces instructions.

La combinaison de first/fput avec last/lput devient encore plus intéressante si on modifie une des parties séparées avant la fusion avec fput ou lput.

Modification des éléments

```
to DADA :LISTE
op fput (word first :LISTE "DADA ) (bf :LISTE)
end
```

Appel: pr DADA [COMMENT CA VA]

Réponse: COMMENTDADA CA VA

Les parenthèses ne sont là que pour faire joli et vous pouvez fort bien les négliger. DADA ajoute la chaîne de caractères DADA au premier mot de la phrase.

Si au lieu de cela c'est le mot lui-même qui est rajouté, on obtient un effet de bégaiement:

```
to BEGAIEMENT :LISTE
op fput (word first :LISTE first :LISTE) bf :LISTE
end
```

Appel: pr BEGAIEMENT [COMMENT CA VA]
Réponse: COMMENTCOMMENT CA VA

Pour bégayer vraiment, il faut redoubler tous les mots. Si vous examinez comment nous sommes passés de A.LA.FIN à la procédure RENVERSER, vous comprendrez comment BEGAIEMENT doit être complété.

Le reste produit par bf :LISTE doit lui-même être soumis à l'opération de bégaiement. Dans le texte du programme, il faut donc remplacer:

bf :LISTE par BEGAIEMENT bf :LISTE

Nous déclenchons ainsi un bégaiement pour tous les mots. Il nous faut cependant encore prévoir une interruption de la répétition. Cette interruption pourra être tout à fait semblable à celle de RENVERSER.

```
to BEGAYER :LISTE
if emptyp :LISTE [op []]
op fput (word first :LISTE first :LISTE) (BEGAYER bf
:LISTE)
end
```

Appel: pr BEGAYER [COMMENT CA VA]

Réponse: COMMENTCOMMENT CACA VAVA]

9.9 Déterminer de quoi il s'agit: les mots de test

emptyp est un mot de test LOGO. emptyp :a a pour résultat TRUE ou FALSE, exactement comme une opération de comparaison du type :a > :b.

Les mots de test se terminent par p dans le vocabulaire de base LOGO mais les mots LOGO terminés par p ne sont pas nécessairement des mots de test. Lors de l'appel de ces mots, l'entrée elle-même n'est pas modifiée. Ces mots LOGO sont utilisés pour tester les propriétés de certaines situations de façon à pouvoir prendre des décisions sur l'emploi ultérieur de ces situations. emptyp est très souvent employé pour programmer l'interruption des répétitions comme c'était d'ailleurs le cas dans les exemples précédents.

D'autres mots de test permettent de déterminer le type d'un objet:

- numberp :a détermine si la valeur de a est un nombre.

- wordp :a examine si l'entrée est un mot.

- listp :a examine si la valeur de a est une liste.

D'après les règles du LOGO, un nombre est donc aussi un mot.

```
numberp 5
wordp 5
listp 5
```

Réponses: TRUE, TRUE, FALSE.

Le mot de test listp recherche en quelque sorte les crochets extérieurs, le nombre d'éléments ne jouant aucun rôle ici. Un mot unique placé entre crochets ne sera pas considéré comme un mot mais bien comme une liste:

```
listp [UNE LISTE]
listp [MOT]
listp "MOT
listp []
```

Réponses: TRUE, TRUE, FALSE, TRUE.

Un autre mot de test très utile:

```
memberp
```

memberp permet de déterminer si un mot ou une liste fait partie d'une liste donnée ou bien si un caractère apparaît dans un mot.

```
memberp "BRUIT [LES PAROLES SONT DU BRUIT ET DE LA FUMEE]
memberp "VILAIN [LE TEMPS EST BEAU CETTE SEMAINE]
memberp "A "ABC
memberp [UNE] [VOICI [UNE] LISTE IMBRIQUEE]
memberp ". "5.16
```

Le second exemple donnera FALSE, tous les autres TRUE.

> memberp teste si la première entrée est contenue dans la seconde.

Dans la hiérarchie caractère, mot, chapitre, livre, ... la première entrée doit se situer à un niveau inférieur à celui de la seconde valeur en entrée.

DR LOGO pousse même l'obligeance jusqu'à vous dire en quelle position le caractère ou l'élément de liste recherchés ont été trouvés. Il suffit pour cela de poser la question Où? (where).

```
memberp "BRUIT [LES PAROLES SONT DU BRUIT ET DE LA FUMEE] where
```

La réponse positive TRUE sera suivie de l'indication de la position 5. Avec des caractères isolés, where et memberp fonctionnent exactement de la même façon:

```
(pr memberp "B "ABC where)
```

Réponse: TRUE 2

Si le caractère ou l'élément de liste recherchés n'ont pas été trouvés, where sortira zéro comme position:

```
(pr memberp "D "ABC where)
```

Réponse: FALSE 0

piece vous permet d'extraire des portions de listes ou de chaînes de caractères à condition que vous connaissiez les positions de départ et de fin. where vous permet de définir des parties de listes ou de chaînes de caractères d'après le premier ou le dernier élément.

```
to DEBUT :objet :but
if memberp :objet :but [op piece 1 where :but] !
[op :but]
end
```

Exemples:

DEBUT "BRUIT [LES MOTS SONT DU BRUIT ET DE LA FUMEE]

Réponse: [LES MOTS SONT DU BRUIT]

DEBUT "X "MOT

Réponse: MOT

La procédure début sort le début d'une liste ou d'un mot jusqu'à l'élément de liste ou jusqu'au caractère objet.

Si l'objet n'est pas trouvé, le résultat sera constitué par la liste entière ou par la chaîne de caractères entière. Si on veut sortir dans ce cas une liste vide ou un mot vide, il nous faut encore tester si but est un mot ou une liste, de façon à ce que le résultat soit du bon type.

```
to debut2 :objet :but
if memberp :objet :but [op piece 1 where :but]
if wordp :but [op "] [op []]
end
```

Exemples:

DEBUT2 "TONNELIER [LES MOTS SONT DU BRUIT ET DE LA FUMEE]

Réponse: []

```
DEBUT "X "MOT
```

Réponse:

Vous avez certainement remarqué que, dans les deux cas, nous avons utilisé le mot LOGO if pour deux possibilités alors qu'à la leçon 7 nous ne présentions qu'une seule liste à exécuter. Vous comprenez bien sûr ce que cela signifie:

- Avec if, l'expression logique à tester est d'abord suivie de la liste d'instructions LOGO qui devront être exécutées si le résultat est TRUE.

- Si on ajoute une seconde liste d'instructions, celle-ci sera exécutée si le résultat est FALSE. Si cette seconde liste est vide, elle peut être négligée.

Tirage du loto: un autre exemple avec memberp

Nous vous avons déjà présenté des résultats de loto (6 chiffres parmi 49) dans le premier chapitre sur le calcul sous LOGO. Il pouvait alors se produire que des nombres déjà tirés au sort se répètent avant que tous les chiffres aient été tirés. Pour éviter des doublets, il faut absolument que les résultats du tirage soient conservés pour que les chiffres suivants puissent leur être comparés. Avec memberp, il sera très facile de tester ensuite si une répétition s'est produite, auquel cas il faudra tirer d'autres nombres au hasard.

```
to LOTO :chiffres
local "nombre
if count :chiffres = 6 [op :chiffres]
make "nombre (1+random 49)
if memberp :nombre :chiffres [op LOTO :chiffres]
op LOTO lput :nombre :chiffres
end
```

La procédure LOTO s'appelle elle-même en deux endroits et débouche, ce sur quoi nous attirons votre attention, sur une valeur en sortie. C'est pourquoi l'auto-appel doit se faire avec op LOTO.
Si le mot de test memberp débouche sur la réponse Oui, c'est-à-dire si le nombre tiré n'est plus dans la machine de tirage au sort, une répétition du tirage est déclenchée par auto-appel. Si le nombre n'avait pas encore été tiré, la liste chiffres qui contient les nombres déjà tirés est complétée avec lput. Le nombre tiré doit être noté avec make car une répétition avec la fonction random entraînerait un autre résultat.

L'appel devient plus commode si on utilise un programme-cadre:

```
to TIRAGE
make "chiffres LOTO []
(pr [RESULTAT:] BL :chiffres [-nombre complementaire:] last !
:chiffres)
end
```

Exemple:

```
TIRAGE

RESULTAT: 2 27 44 49 46 -nombre complementaire: 6
```

Après un tirage du loto, les nombres sortis sont présentés triés. Comme DR LOGO sur les ordinateurs AMSTRAD ne dispose pas d'instruction de tri intégrée, le tri peut être réalisé tout simplement en examinant pour chaque boule si elle a déjà été tirée ou non. L'ordre qui convient sera établi lors de cet examen.

```
to TRI :i :l :e;trier des nombres de 1 à i
repeat :i [if memberp :i :l [make "e se :i :e] make "i !
:i - 1] op :e
end
to TIRAGE
make "chiffres LOTO []
(pr [RESULTAT: ] TRI 49 bl :chiffres [] [- nombre !
complementaire:] last :chiffres)
end
```

9.10 Opérations de comparaison

Pour tester si deux objets sont identiques, on peut utiliser le mot de test equalp.

```
make "liste [LES PAROLES SONT DU BRUIT ET DE LA FUMEE]
make "phrase [NOUS SOMMES MERCREDI]
make "copie :liste

equalp :liste :phrase                Réponse: FALSE
equalp :liste :copie                 Réponse: TRUE
equalp 10 20                         Réponse: FALSE
equalp "mot "mot                     Réponse: TRUE
```

equalp peut être appliqué à des listes, à des mots ou à des nombres. Le mot de test equalp suit la syntaxe habituelle des mots de test. Ce mot peut toutefois également être simplement remplacé par un signe égale:

```
= :liste :phrase                     Réponse: FALSE
= "mot "mot                          Réponse: TRUE
```

Le signe égale présente également l'avantage de pouvoir être non seulement employé avec la notation préfixée, comme dans les exemples précédents, mais aussi avec la notation infixée:

```
:liste = :phrase                     Réponse: FALSE
"mot = :phrase                       Réponse: FALSE
```

On peut tester s'il y a différence au moyen de la négation de l'égalité. C'est à cet effet que presque tous les langages de programmation disposent du mot not:

```
not equalp :liste :phrase            Réponse: TRUE
not = "mot "mot                      Réponse: FALSE
```

Dans la notation infixée, la différence peut aussi être testée avec les signes inférieur à et supérieur à accolés:

```
:liste <> :phrase                    Réponse: TRUE
```

Les opérations de comparaison 'supérieur à' et 'inférieur à' (> ou <) ne peuvent être appliquées qu'aux nombres et aux mots.

> "LOGO "BASIC	Réponse: TRUE
"HELENE < "JEANNE	Réponse: TRUE
10 > 20	Réponse: FALSE
< 1E10 1E15	Réponse: TRUE

La comparaison se fait pour les mots d'après l'ordre alphabétique. Un mot sera dit plus grand qu'un second s'il figure plus loin dans un dictionnaire que le premier, selon l'ordre alphabétique.

9.11 Mélanger des listes

Vous savez déjà comment random permet de faire entrer le hasard en jeu. Mais vous pouvez aussi mélanger au hasard des mots entiers ou des listes entières. C'est à cela que sert le mot LOGO shuffle.

```
shuffle :liste
```

Essayez avec CONTROL+Y ce qui se passe si vous répétez votre entrée! Les mots sont présentés chaque fois dans un ordre totalement arbitraire. shuffle ne peut être appliqué qu'à des listes mais celles-ci peuvent aussi être imbriquées.

```
make "BOITE !
[CECI [EST UNE [LISTE]] [TRES] IMBRIQUEE]
shuffle :BOITE
```

Réponse: [IMBRIQUEE [EST UNE [LISTE]] CECI [TRES]]

(Il ne s'agit bien sûr là que d'un exemple de résultat possible. Le hasard peut faire que celui que vous obtiendrez soit très différent.)

shuffle peut aussi permettre de représenter un tirage du loto. On peut ainsi, d'une façon similaire, mélanger 49 boules et sélectionner 7 boules.

```
to BOULES :n :tambour
if :n=0 [op :tambour]
boules :n-1 fput :n :tambour
end
```

Appel: BOULES 49 []

Le résultat de BOULES sera une liste des nombres de 1 à 49. Lors de l'appel, on commencera avec la liste vide. Dans la troisième ligne, un nombre sera rajouté à gauche avec fput. La répétition se fait de manière récursive, le nombre n rajouté diminuant chaque fois de 1 jusqu'à ce qu'on atteigne la valeur zéro qui entraîne la fin de la récursion. Si vous remplissez un tambour de boules avec:

```
make "tambour BOULES 49 []
```

l'opération de tirage au sort pourra être effectuée avec:

```
piece 1 7 shuffle :tambour
```

9.12 Les listes de propriétés

Un objet est défini par son nom et ses propriétés. En ce qui concerne les propriétés, il nous faut distinguer entre le concept, par exemple la couleur et la valeur correspondante, par exemple rouge. Une propriété se compose donc d'une paire comprenant la propriété véritable et sa valeur. Le nom de la propriété est un mot, la valeur correspondante peut être un mot ou aussi une liste.

Les listes de propriétés sont des listes composées de paires de ce type.

Elles constituent un concept très général et sont la base de l'interpréteur LOGO. Il est déjà apparu plusieurs fois que les données et les programmes sont traités en LOGO de façon analogue puisque ce sont toujours des objets avec des noms et des propriétés.

Pour savoir quelles propriétés se cachent derrière un nom, on

utilise le mot LOGO plist.

```
make "NOMBRE 10
plist "NOMBRE
```

Réponse: [.APV 10]

Le premier élément d'une paire est le nom de la propriété, ici .APV (pour associated property value). Les propriétés commençant par un point sont des propriétés relevant du système. Le manuel livré avec votre machine vous en énumère six. Notez que ces propriétés doivent absolument être écrites en majuscules.

La propriété système .APV indique la valeur d'une variable globale.

Il y a certainement encore des procédures disponibles dans la mémoire de travail. Demandez les propriétés de certaines procédures avec plist.

```
plist "NOMDEPROCEDURE
```

Vous trouverez alors la propriété .DEF dont la valeur correspondante est le texte du programme sous la forme d'une liste. Vous trouverez peut-être également d'autres propriétés qui indiquent le formatage du texte ou des commentaires. Mais nous ne parlerons pas de ces propriétés.

Essayez aussi:

```
plist "fd
```

Réponse: [.PRM 7020]

La propriété .PRM indique l'appartenance au vocabulaire de base de DR LOGO. La valeur numérique à la suite de cette indication n'est pas exploitable, elle n'a de signification que de façon interne, pour l'interpréteur.

La valeur d'une propriété peut être lue à l'aide de gprop.

```
gprop "nombre ".APV
```

Réponse: 10

```
gprop "NOMDEPROCEDURE ".DEF
```

Réponse: le texte du programme

```
gprop "nombre "couleur
```

Réponse: []

Si la liste demandée n'existe pas, une liste vide vous est renvoyée.

Le mot LOGO pprop permet d'affecter une propriété à un objet.

```
pprop "AUTO "COULEUR "ROUGE
pprop "AUTO "PRIX "60000FF
pprop "AUTO "MARQUE "AUCEDES
plist "AUTO
```

Réponse: [MARQUE AUCEDES PRIX 60000FF COULEUR ROUGE]

```
gprop "AUTO "COULEUR
```

Réponse: ROUGE

Le mot LOGO remprop permet de supprimer des propriétés. Lorsqu'une propriété est supprimée, sa valeur ne change pas mais c'est toute la propriété qui est supprimée.

```
remprop "AUTO "PRIX
```

Dans notre exemple, l'auto n'aura plus de prix. Une modification de la valeur est obtenue avec pprop. On peut aussi déterminer à

l'inverse, grâce à l'instruction glist, quels noms possèdent une propriété déterminée.

```
pprop "CIEL "COULEUR "BLEU
glist "COULEUR
```

Réponse: [AUTO CIEL]

Il existe enfin encore un mot LOGO pps qui permet de sortir tous les noms avec les propriétés correspondantes, sans tenir compte des propriétés système.

```
pps
```

Réponse: CIEL's COULEUR is BLEU
AUTO's COULEUR is ROUGE ...

9.13 Remplacement avec listes de propriétés

Supposez qu'il s'agisse d'imprimer des petites annonces pour véhicules d'occasion. Le texte de l'annonce possédera une forme standard de sorte que seules certaines propriétés particulières d'un véhicule telles que la marque, l'âge et le prix devront être changées pour chaque annonce.

```
make "annonce [[Occasion à saisir:] [MARQUE TYPE Année ANNEE] [Couleur: COULEUR] [Prix: PRIX FF] [Garage Meunier Alfortville]]
```

En entrant cette longue ligne pour définir la valeur de annonce, une faute de frappe peut bien sûr facilement vous échapper. Si vous remarquez l'erreur immédiatement, la ligne d'entrée peut être à nouveau placée sur l'écran avec CONTROL+Y ou également avec COPY sur le PCW. Vous pourrez alors la corriger. Si cela n'est plus possible, la variable annonce peut être traitée au moyen de l'éditeur. Après avoir entré

```
ed "annonce
```

vous verrez apparaître la ligne en mode d'édition et vous pourrez la traiter comme un texte de programme. Vous devrez ici aussi sortir de l'éditeur avec CONTROL+C.

```
pprop "auto "MARQUE "Aucedes
pprop "auto "TYPE "100GL
pprop "auto "COULEUR "rouge
pprop "auto "PRIX 20000
pprop "auto "ANNEE 1983
```

Nous avons ainsi décrit les propriétés d'un véhicule dont nous aurons besoin pour les insérer dans le texte de l'annonce à la place des variables. Le programme REMPLACE se chargera de cela pour une phrase unique à condition que les propriétés nécessaires soient déjà disponibles dans la liste de variables.

```
make "eliste plist "auto

to REMPLACE :phrase :nouveau
if emptyp :phrase [op nouveau]
local "mot make "mot first :phrase
if memberp :mot :eliste [op REMPLACE bf :phrase se !
:nouveau item 1+where :eliste] [op REMPLACE bf :phrase !
se :nouveau :mot]
end
```

Comme REMPLACE ne traite dans un premier temps que des phrases isolées, les phrases doivent être extraites avec item du texte de l'annonce pour pouvoir être testées.

```
REMPLACE item 2 :annonce []
```
Réponse: [Aucedes 100GL Année 1983]

```
REMPLACE item 1 :annonce []
```
Réponse: [Occasion à saisir:]

La procédure REMPLACE a deux entrées: la phrase à traiter et la nouvelle phrase qui en résultera et qui se présente lors de l'appel tout d'abord sous forme d'une liste vide. Le mot de test memberp permet de déterminer si le mot actuellement examiné est contenu

dans la liste de propriétés de auto. Si c'est le cas, alors la valeur de la propriété figurera comme prochain élément de eliste, la position pouvant être déterminée à l'aide de where.

La phrase est maintenant analysée mot pour mot. L'appel récursif permet ici de raccourcir chaque fois la première entrée d'un mot et un mot est ajouté chaque fois à la suite de la nouvelle phrase en train de se construire. Comme REMPLACE renvoie un résultat, l'auto-appel doit également commencer par op.

```
to FPARAGRAPHE :paragraphe :objet
(local "eliste "i)
make "eliste plist :objet make "1
repeat count :paragraphe [pr REMPLACE item :i !
:paragraphe [] make "i :i+1]
end
```

Appel: FPARAGRAPHE :annonce "auto

Dans FPARAGRAPHE un paragraphe entier, c'est-à-dire une liste de phrases, est traité avec REMPLACE. La répétition est ici effectuée avec repeat; une variable de comptage i est utilisée pour détacher chaque fois la i-ième phrase. La répétition aurait tout aussi bien pu être formulée de façon récursive.

Bien entendu, c'est tout de même assez pénible de devoir entrer ici les propriétés du véhicule l'une après l'autre avec pprop. On peut utiliser pour cela le programme suivant qui utilise déjà l'instruction rq pour une entrée au clavier, instruction qui sera expliquée plus en détail au chapitre suivant.

```
to DAUTO
type "Marque: pprop "auto "MARQUE rq
type "Type: pprop "auto "TYPE rq
type "Couleur: pprop "auto "COULEUR rq
type "Prix: pprop "auto "PRIX rq
type "Année: pprop "auto "ANNEE rq
end
```

Après l'entrée de DAUTO, les propriétés nécessaires seront

demandées sur l'écran et la réponse de l'utilisateur devra se terminer chaque fois par Return. Les propriétés peuvent par ailleurs également contenir des espaces, par exemple "bleu métallique" pour la couleur.

Exercices

1) Complétez les programmes pour produire des phrases aléatoires. Ecrivez les procédures ADVERBE et COMPLEMENT.LIEU pour obtenir aussi pour ces parties de la phrase des sorties variables.

2) Ecrivez une procédure POINTE qui enlève à chaque étape un lettre à l'avant aussi bien qu'à l'arrière du mot entré.

3) Une procédure devra tester si l'entrée constitue un palindrome, c'est-à-dire un mot pouvant être lu de manière identique aussi bien de droite à gauche que de gauche à droite (par exemple OTTO). La procédure devra sortir soit le mot "TRUE soit le mot "FALSE.

4) Formulez une action LOGO qui mélange arbitrairement les lettres du mot entré.

5) Nous écrivons normalement les fractions décimales avec une virgule. Le programme devra écrire le nombre avec virgule à l'américaine. (Avec la virgule, le nombre fourni en valeur d'entrée doit être écrit précédé de guillemets, "1,57 par exemple. En anticipant sur la prochaine leçon, essayez entre autre first rq comme entrée!)

6)
```
to CONVERTIR :n :b; convertir n du système décimal au
système de base b
if :n = 0 [op "]
op word (CONVERTIR quotient :n :b :b) (remainder :n :b)
end
```

Etudiez ce programme! Si la base est supérieure à 10, on utilise des lettres pour représenter les chiffres suivants

(10=a, 11=b, etc...). Généralisez CONVERTIR en prévoyant ces cas, notamment pour la base 16.

7) Vous êtes peut-être un joueur au loto superstitieux? Modifiez les procédures de tirage du loto de façon à ce que le 13 ne puisse jamais sortir.

8) shuffle ne peut être appliqué à des mots ni, de ce fait, à des nombres. Pour pouvoir mélanger un mot ou un nombre avec shuffle, le mot, ou le nombre, doit d'abord avoir été décomposé en caractères et ces caractères doivent avoir été réunis dans une liste. Ecrivez donc deux procédures, l'une qui transforme un mot en une liste composée des caractères de ce mot et l'autre qui recompose un mot à partir de la liste mélangée.

Leçon 10: Entrées et sorties

Nouveaux éléments de vocabulaire:
rq, rl, rc, keyp, ascii, char, go, label, wait

Programmes:
SALUTATION, CERCLE, REACTION

10.1 Listes d'entrées

Dès que votre ordinateur est allumé, il consomme de l'électricité. Il n'en consomme certainement pas autant qu'un appareil de chauffage mais vous êtes tout de même en droit d'attendre qu'il se passe quelque chose. Lorsque le système LOGO n'est pas en train d'exécuter vos instructions, il attend une entrée qui sera normalement effectuée au clavier. C'est pourquoi un point d'interrogation est sorti sur l'écran et pourquoi vous voyez le curseur. Pour l'ordinateur, cela constitue une action au même titre que la résolution de problèmes de calcul.

Mais il vous est également possible de placer l'ordinateur dans cet état d'attente d'une entrée au moyen des mots LOGO rl et rq. Examinons, à titre exemple, un programme simple:

```
to SALUTATION
pr [COMMENT VOUS APPELEZ-VOUS?]
make "NOM rl
(pr [AI-JE BIEN COMPRIS, VOUS VOUS APPELEZ] :NOM "?)
if rq = "NON [SALUTATION] [(pr [D'ACCORD, BONJOUR ]
:NOM)]
end
```

Après appel de SALUTATION, la ligne de texte:

```
COMMENT VOUS APPELEZ-VOUS?
```

sera sortie et le curseur apparaîtra dans la ligne suivante de l'écran. Mais vous ne voyez pas apparaître le point d'interrogation qui est normalement placé devant le curseur lorsque le LOGO attend de nouvelles instructions, ce qu'on appelle 'l'interrogation' LOGO. Le système attend maintenant une entrée tout en se trouvant dans la procédure SALUTATION. Vous pouvez vous en convaincre en interrompant l'exécution avec STOP (ESC). Le LOGO vous répondra:

```
Stopped! in SALUTATION: ...
```

Après que vous ayez entré une suite de caractères, le dialogue se poursuit.

> rl place le système dans un état d'attente. Le programme ne se poursuit qu'après que vous ayez appuyé sur la touche RETURN.

rl peut être également utilisé en mode direct, avec une réponse du type:

```
[...]
```

Les crochets indiquent que rl sort une liste comme résultat. Cela peut aussi être aisément confirmé avec le mot de test listp:

```
listp rl
```

> Réponse, quelle que soit l'entrée: TRUE

Le résultat de rl peut aussi être traité avec des mots LOGO agissant sur les listes, par exemple:

```
count rl
first bf rl
```

Cela vous ouvre la voie au vaste domaine des programmes de dialogue avec entrée.

Dans l'avant-dernière ligne de SALUTATION, vous trouvez le mot LOGO rq, abréviation de rq. Ce mot se distingue de rl par le

fait qu'il attend non pas une liste mais un mot.

```
wordp rq
```

N'importe quelle entrée débouchera sur la réponse TRUE, même si vous entrez plusieurs mots séparés par des espaces ou si vous placez une phrase entre crochets. Avec rq, les espaces aussi bien que les crochets seront toujours interprétés comme des caractères normaux faisant partie du mot et non, comme d'habitude, comme des marques de séparation. Vous pouvez d'ailleurs le vérifier en faisant compter le nombre d'éléments avec count.

> rq renvoie les caractères tapés sous la forme d'un mot. L'ordinateur indique avec le curseur qu'il est prêt à recevoir une entrée. L'entrée n'est considérée comme terminée qu'après que vous ayez appuyé sur la touche RETURN.

10.2 Sauts en avant et en arrière

Dans le programme SALUTATION, nous avons déjà prévu une certaine forme de contrôle de l'entrée au clavier. L'utilisateur doit confirmer lui-même son entrée, le nom en l'occurrence. S'il n'y a pas confirmation mais au contraire la réponse NON, la procédure SALUTATION est appelée de façon récursive. Un programmeur prévoyant doit toujours prendre en compte de possibles erreurs d'entrée en faisant contrôler, dans le programme même, la validité de toute entrée produite avec rl ou rq, avant de passer au traitement de cette entrée.

Examinons, à titre d'exemple, un programme de calcul de cercle:

```
to CERCLE
pr [CALCUL DE CERCLE]
pr [QUELLE DOIT-ETRE LA TAILLE DU RAYON?]
make "r rq
(pr [PERIMETRE =] 6.283185*:r)
(pr [SURFACE=] 3.14159*:r*:r)
end
```

L'entrée du rayon est effectuée avec rq dans la quatrième ligne. Il peut cependant arriver que l'utilisateur tape par erreur autre chose qu'un nombre. Lorsque le programme essaiera ensuite d'exécuter cet exemple de calcul, il en résultera alors nécessairement un message d'erreur. Il vaudrait donc mieux contrôler si on a bien affaire à une valeur numérique. Le mot de test correspondant est numberp.

Si ce n'est pas un nombre qui a été entré, on ne passera pas au calcul du cercle mais on demandera une nouvelle entrée. La condition correspondante devrait donc se présenter ainsi: 'si pas un nombre, alors ...'. La négation d'un mot de test s'obtient avec le mot LOGO not:

```
if not numberp [...]
```

Que faut-il mettre à la place des points de suspension? Le programme doit revenir à la ligne:

```
pr [QUELLE DOIT-ETRE LA TAILLE DU RAYON?]
```

Vous ne connaissez jusqu'à présent que la possibilité d'appeler d'autres actions, d'appeler une procédure de façon récursive ou de mettre fin à une procédure. Il est cependant également possible de revenir, à l'intérieur d'un programme, à des lignes de programmes ou de parvenir à des lignes placées plus loin dans le programme en sautant certaines lignes de programme. C'est go qui permet d'effectuer de tels sauts à l'intérieur d'une procédure.

```
to CERCLE
pr [CALCUL DE CERCLE]
label "ENTREE pr [QUELLE DOIT-ETRE LA TAILLE DU RAYON?]
make "r rq
if not numberp :r [go "entree]
(pr [PERIMETRE =] 6.283185*:r)
(pr [SURFACE=] 3.14159*:r*:r)
end
```

Le but du saut doit être marqué pour qu'un saut soit possible. Pour cela, il faut placer le mot LOGO label au début de la ligne de

programme qui sera le but du saut puis, à la suite de ce mot, un nom marquant la ligne. Cette marque n'acquerra de signification que lors du traitement de l'instruction go.

Presque tous les langages de programmation possèdent des instructions de saut. Celles-ci sont même très fréquentes au niveau de ce qu'on appelle le langage-machine. L'emploi de go n'est pas vraiment caractéristique du LOGO car on dispose en LOGO de l'outil beaucoup plus puissant que constitue l'appel de procédure avec la possibilité de l'appel récursif. Cependant, les sauts ne peuvent pas se faire n'importe où en LOGO mais seulement à l'intérieur d'une même procédure. Comme les procédures LOGO typiques ne comportent que relativement peu de lignes, les sauts ne se font que par dessus un petit nombre de lignes.

La clarté de la structure d'un programme est de ce fait peu affectée par l'instruction go. Il serait cependant tout à fait insensé de formuler de longues procédures avec toute une série de buts de saut et d'instructions go. Un langage de programmation tel que le LOGO, aussi puissant que le LOGO, qui est axé de façon cohérente sur la solution de problèmes, n'est pas conçu pour réaliser des structures de programmes primitives, proches de la machine.

Test de réaction

Nous allons développer un programme qui teste le temps de réaction. On demandera à un candidat pour ce test d'appuyer sur une touche déterminée lorsque le caractère correspondant apparaîtra dans la fenêtre de dialogue. Un tel programme permet aussi d'apprendre à connaître le clavier.

Ecrivons d'abord quelques procédures auxiliaires pour le titre ainsi qu'une autre procédure d'arrêt momentané.

```
to TITRE
ct pr "
pr [TEST DE REACTION] pr "
pr [APPUYEZ AUSSI VITE QUE POSSIBLE] pr "
```

```
pr [SUR LA TOUCHE QUI VA APPARAITRE SUR L'ECRAN ]
pr "
end

to ARRET
repeat 150 []
end
```

ARRET permet de programmer une pause assez longue qui se terminera sans intervention extérieure.

L'affichage du caractère à entrer doit bien sûr suivre dans un intervalle de temps qui ne puisse pas être calculé à l'avance. Nous pouvons utiliser pour cela des nombres aléatoires avec lesquels nous construirons une boucle d'attente qui sera parcourue autant de fois que fixé par le hasard.

```
repeat random 300 []
```

DR LOGO sur les ordinateurs CPC AMSTRAD, mais donc pas sur le PCW, dispose d'un mot spécial pour de telles boucles d'attente.

```
wait 6000
```

Le déroulement du programme sera ainsi interrompu pendant environ 100 secondes. La ligne repeat de ARRET peut être réécrite de la façon suivante, avec wait:

```
wait random 500
```

Vous pouvez naturellement aussi choisir une autre valeur en entrée pour random.

Pour pouvoir déterminer si une touche a été appuyée, le LOGO dispose du mot de test:

```
keyp
```

Après qu'une touche ait été appuyée, keyp fournit la valeur TRUE, sinon FALSE.

Dans le test de réaction, le contrôle de la frappe des touches doit bien sûr permettre de mesurer la vitesse de réaction. Cela sera réalisé dans la procédure TOUCHE:

```
to TOUCHE :t
local "I make "I 0
label "DEBUT
if keyp [make "ta] [rc make "I :I+1 go "DEBUT]
if ascii :ta = :t [op :I] [op -1]
end

to EVALUER :I
if :I < 0 [op [UNE ERREUR !]]
if :I < 3 [op [EXCELLENTE !]]
if :I < 6 [op [TRES BONNE !]]
if :I > 15 [op [LENTE !]] [op [BONNE !]]
end
```

TOUCHE ordonne une répétition de l'interrogation du clavier avec go à la marque de saut DEBUT. Le nombre de répétitions est compté avec la variable locale I puis sorti comme résultat. Le résultat de TOUCHE est une mesure de la rapidité. Sa valeur peut ensuite être évaluée dans la procédure EVALUER.

Avant de poursuivre plus avant l'explication de TOUCHE, il nous faut encore présenter le programme principal, REACTION.

```
to REACTION
TITRE
repeat random 300 []
if keyp [(pr "DISQUALIFIE rc!
[VOUS AVIEZ DEJA APPUYE!]) ARRET REACTION]
make "t 97+random 26 pr char :t
pr " (type [VOTRE REACTION A ETE] EVALUER TOUCHE :t)
pr " ARRET REACTION
end
```

Dans la quatrième ligne de REACTION, on contrôle si vous n'avez pas frappé une touche trop tôt, avant qu'on ne vous l'ait demandé. Si c'est le cas, vous êtes disqualifié. Le mot LOGO rc est alors

utilisé pour lire le caractère correspondant à la touche appuyée précocément. Ce caractère est sorti comme résultat.

Une réponse positive du mot de test keyp ne signifie pas qu'on est précisément en train d'appuyer sur une touche, elle signifie simplement qu'il y a une entrée au clavier qui n'a pas encore été traitée par le LOGO. Seule la lecture avec rc effacera cette indication. C'est pourquoi il nous faut lire le caractère même en cas d'entrée précoce. ARRET provoque une pause artificielle et l'auto-appel avec REACTION fait tout recommencer. La lettre à frapper est sélectionnée dans la cinquième ligne du programme principal REACTION.

```
make "t 97+random 26
```

Un nombre entier entre 97 et 122 est ainsi tiré au hasard et rangé sous le nom t. La valeur de t est alors transmise à la procédure TOUCHE. La relation entre cette valeur numérique et une lettre de l'alphabet est établie avec:

```
pr char :t
```

Avec char entre ici en jeu une nouvelle fonction. Pour la comprendre, le mieux est de la relier à la fonction qui apparaît dans la dernière ligne de TOUCHE:

```
ascii rc
```

- char affecte un caractère déterminé à un nombre.

- Inversement, ascii affecte un nombre à un caractère.

Exemples:

char 97	Réponse: a
char 122	Réponse: z
char 65	Réponse: A
char 90	Réponse: Z
ascii "B	Réponse: 66

```
ascii "q              Réponse: 113

ascii char 97        Réponse: 97
char ascii "z        Réponse: z
```

Les caractères sont des mots pour le LOGO. L'affectation de nombres à des caractères permet de les coder. char et ascii se réfèrent, en l'occurrence, au code standard ASCII (American Standard Code for Information Interchange). Les deux derniers appels montrent que char et ascii s'annulent réciproquement.

Notez bien que les majuscules et les minuscules n'ont pas les mêmes valeurs pour le code ASCII. Dans le cas du programme REACTION décrit ici, le plus simple serait de n'utiliser que des minuscules. Le jeu de caractères français pose un problème à cet égard car il n'est pas pris en compte par le code ASCII. Les ordinateurs CPC ont un clavier ASCII et ils travaillent normalement sans les accents français. Le PCW AMSTRAD a bien un clavier français mais il vaut mieux sélectionner le jeu de caractères américain lorsqu'on veut travailler sous DR LOGO.

Vous pouvez bien sûr utiliser également la plupart des autres touches pour le test de réaction. Le moyen le plus simple pour déterminer le code ASCII correspondant est d'entrer la ligne

```
ascii rc
```

L'effet de TOUCHE peut être aisément constaté par des exemples:

```
TOUCHE 97
```

Le programme TOUCHE attend la frappe d'une touche, sans qu'un curseur apparaisse sur l'écran. L'attente est programmée dans la première ligne de TOUCHE. Si le résultat de keyp est FALSE, c'est-à-dire si aucune touche n'a été enfoncée, alors la valeur de la variable de comptage I est augmentée de 1. Si vous tapez maintenant la touche de lettre a, qui correspond au code ASCII 97, vous verrez par exemple apparaître:

```
Réponse: 85
```

Le nombre indique combien de fois la première ligne de TOUCHE a été parcourue. Si, en effet, le mot de test keyp indique, dans la première ligne de TOUCHE, qu'une touche a effectivement été frappée, alors rc détermine le caractère frappé et le range sous le nom de TA. La boucle de répétition de la troisième ligne est interrompue et on teste maintenant, dans la quatrième ligne, si le code ASCII de la touche frappée est conforme à la valeur transmise par la seconde variable en entrée t. S'il y a identité, c'est-à-dire, en l'occurrence, si vous avez bien appuyé sur la touche correcte, a, le nombre I des répétitions sera sorti comme résultat. Ce nombre représente une mesure du temps d'attente. Si par contre aucune conformité n'est constatée, une message d'erreur apparaîtra. Si vous appuyez donc sur une touche autre que a, b par exemple, TOUCHE réagira ainsi:

```
Réponse: -1
```

La valeur -1 renvoyée comme résultat vous indique que b n'est pas la bonne touche.

Format de l'écran

Les sorties sur l'écran apparaissent là où le curseur se trouve actuellement et nous avons jusqu'ici laissé le système LOGO se débrouiller tout seul. En règle générale, le curseur est fixé après chaque sortie au début de la prochaine ligne libre de l'écran.

Dans la procédure TITRE, nous sommes intervenu dans la structure de l'écran avec ct. L'écran de texte était en effet vidé et le curseur était placé dans le coin supérieur gauche de l'écran. Le curseur peut être positionné librement sur l'écran à l'aide du mot LOGO setcursor.

```
setcursor [20 12]
```

Le curseur sera ainsi placé dans la colonne 20 de la douzième ligne. Si vous avez entré cette ligne en mode direct, le point d'interrogation apparaîtra au centre de l'écran avec, à sa suite, le curseur clignotant. On peut inversement demander avec cursor la

situation actuelle du curseur

```
cursor
```

Résultat: [0 13]

Le résultat de cursor est une liste composée de deux nombres. Le premier nombre indique comme pour setcursor le numéro de colonne et le second nombre le numéro de ligne de la position du curseur lors de l'appel. La position renvoyée indique où se produira ou bien où devrait se produire (si vous n'intervenez pas) la prochaine sortie sur l'écran. Notez que le résultat est une liste. Comme le LOGO n'autorise jamais qu'une valeur en sortie pour les procédures, de même qu'une fonction mathématique fournit une valeur numérique unique, les deux indications de la situation du curseur doivent être réunies dans une liste pour former un objet unique. Par conséquent, si vous voulez obtenir uniquement le numéro de ligne, vous devez décomposer la liste de sortie:

```
last cursor
```

Pour le numéro de colonne, vous utiliseriez de même first. C'est de la même façon que vous devez procéder pour les procédures que vous aurez vous-même programmées lorsque plusieurs résultats doivent être renvoyés.

- setcursor a besoin en entrée d'une liste de deux valeurs. La première indique le numéro de colonne (0 à 39), la seconde le numéro de ligne (0 à 24).

- cursor ne demande aucune entrée. Le résultat fourni est la position du curseur sous forme d'une liste de deux éléments.

La gestion des renseignements concernant des personnes constitue une sphère d'application importante des ordinateurs qui n'est d'ailleurs pas sans poser de problèmes. Nous n'évoquerons ici qu'une fonction très partielle. Le programme devra permettre d'entrer le nom, le prénom et l'adresse d'une personne. Un tel

programme devra être ergonomique, c'est-à-dire qu'on veillera à ce que l'utilisateur n'ait à effectuer que les frappes de touches vraiment indispensables. Il devra être à cet effet aidé par le programme grâce à des commentaires et à une possibilité de commande du curseur.

Les programmes les plus pratiques se chargent également de contrôler les entrées, de refuser les entrées non autorisées, de permettre des corrections, etc... Nous nous contenterons naturellement ici de vous donner des idées que vous pourrez ensuite développer par un travail de programmation personnel.

Masque d'entrée simple

Dans des programmes de ce type, le guidage de l'utilisateur se fait à travers ce qu'on appelle des masques d'entrée. Cela consiste pour le programme de commande à écrire déjà sur une partie de l'écran. On ne prévoit des blancs que dans des zones bien précises et ce sont ces blancs qu'il s'agira de remplir lors de l'entrée. Le curseur doit pour cela être commandé de telle façon que l'utilisateur ne puisse écrire que dans les blancs prévus et, en général, dans un ordre fixé d'avance. On demandera ici le nom, le prénom, la rue, le numéro, le code postal et la localité.

La façon la plus simple de réaliser un tel masque d'entrée consiste à autoriser exactement une entrée par ligne de l'écran. Si les caractères entrés par l'utilisateur terminent une ligne, l'utilisateur ne peut rien effacer tant qu'il reste dans la ligne.

```
to MASQUE1
ct
setcursor [0 1] pr [Nom:...............]
setcursor [0 3] pr [Prenom:...............]
setcursor [0 5] pr [Rue:...............]
setcursor [0 7] pr [Nro:....]
setcursor [0 9] pr [Code postal:.....]
setcursor [0 11] pr [Ville:..............]
end
```

```
to ENTREE1
MASQUE1
setcursor [4  1] make "Nom rq
setcursor [7  3] make "Prenom rq
setcursor [4  5] make "Rue rq
setcursor [4  7] make "Nro rq
setcursor [12 9] make "Code rq
setcursor [6 11] make "Ville rq
op (se :Nom :Prenom :Rue :Nro :Code :Ville)
end
```

Aucun contrôle des entrées n'est effectué. On pourrait par exemple utiliser le mot de test numberp pour vérifier, pour le code postal et le numéro de la rue, si c'est bien un nombre qui a été entré. Si ce n'est pas le cas, on pourrait répéter la ligne correspondante. Avec les autres indications, on pourrait veiller au contraire à ce que les nombres soient rejetés.

Après avoir appelé

```
ENTREE1
```

le masque programmé apparaît sur l'écran et le curseur est placé dans l'endroit réservé pour le nom. Lorsque l'entrée du nom est terminée avec la touche RETURN, le curseur saute automatiquement à l'endroit prévu pour le prénom, etc... Vous obtenez finalement en réponse par exemple:

Résultat: [Saucisson Jean rue des Frappe-dur 52 42000 Dunkerque]

Pour recevoir toute une liste d'adresses, il faut répéter l'entrée.

```
to ADRESSES :liste
local "personne
label "entree make "personne ENTREE1
if first :personne = "* [op :liste]
make "liste lput :personne :liste
go "entree
end
```

Pour changer un peu, la répétition n'a pas été programmée par récursion, c'est-à-dire par auto-appel. L'entrée des différentes adresses, qui s'effectue dans la procédure ENTREE1, est répétée jusqu'à ce que l'étoile (signe de la multiplication) soit entrée en guise de nom. Pour les autres entrées, on peut tout simplement passer à la suite en appuyant sur la touche RETURN.

Comme la liste d'adresses apparaît comme entrée, la procédure ADRESSES permet de rallonger une liste déjà existante. Lors du premier appel, on entrera une liste vide.

Premier appel: make "association ADRESSES []

Vous entrerez alors les adresses pour plusieurs personnes.

Pour les appels suivants: make "association ADRESSES :association

Masques d'entrée avec rc

L'entrée avec rq est simple à manier parce que la ligne est traitée lors de l'entrée comme une ligne d'entrée normale sous LOGO. Avant de transmettre la ligne d'entrée en appuyant sur la touche RETURN, vous pouvez aller avec les touches de commande du curseur vers la droite et vers la gauche. Vous disposez en outre des possibilités d'édition avec la touche CONTROL. Mais, d'un autre côté, la longueur de la ligne d'entrée ne peut être limitée et l'utilisateur peut fort bien détruire le masque d'entrée fixé par le programme. Il n'est donc pas très conseillé d'utiliser rq lorsque plusieurs entrées doivent être effectuées à l'intérieur d'une même ligne pour un masque d'entrée donné.

La solution consiste à utiliser l'entrée caractère par caractère avec rc qui peut plus facilement être contrôlée. L'inconvénient de cette méthode d'entrée est toutefois que vous devez programmer vous-même la gestion du curseur. rc ne sort pas non plus automatiquement sur l'écran le caractère entré au clavier. Pour un masque d'entrée, les caractères entrés doivent être amenés

spécialement avec type dans les blancs prévus à cet effet. On peut utiliser comme base de ce type d'entrée la procédure ENTREE suivante.

```
to SAISIE :S :Z :M :W
setcursor se :S :Z type char 32
setcursor se :S :Z
(local "c "ac)
label "touche make "c rc make "ac ascii :c
if :ac = 248 [op SAISIE :S-1 :Z :M bl :W]
if :ac = 243 [op :W]
if and :ac > 31 :ac < 123!
[make "W word :W :c type :c] [go "touche]
if :S = :M [op :W]
op SAISIE :S+1 :Z :M :W
end
```

Cette procédure a pour fonction de fixer comme entrée un mot qui, à partir de la position du curseur, qui est fixée par les valeurs d'entrée S et Z (colonne et ligne), pourra s'étendre au maximum jusqu'à la position de colonne M qui est donnée par la troisième valeur en entrée.

Dans la première ligne de SAISIE, type char 32 permet tout d'abord d'effacer la position actuelle du curseur. La seule possibilité de correction prévue est la touche DEL qui correspond au code ASCII 248. La touche RETURN a le code 243 et elle permet de terminer la zone d'entrée dans laquelle on se trouve.

Dans la huitième ligne, on teste si une touche dont le code ASCII est compris entre 32 (espace) et 122 (z) a été enfoncée. Toutes les autres touches seront refusées. Ce test utilise le mot LOGO

```
and
```

qui ne peut être utilisé, dans la version LOGO dont vous disposez, qu'en notation préfixée, c'est-à-dire que le and doit être placé avant les expressions logiques auxquelles il doit être appliqué.

Si le caractère entré est accepté, il est ajouté au mot W en train

de se construire et il est en même temps sorti sur l'écran. Si la position de colonne maximale M est atteinte, l'entrée est également interrompue. On peut ainsi protéger des parties du masque d'entrée qui auront été écrites auparavant contre un effaçage du fait d'une entrée de l'utilisateur du programme. La procédure SAISIE s'appelle finalement elle-même avec une position de colonne augmentée de un.

```
SAISIE 0 10 5 "
```

Le curseur apparaît maintenant au début de la ligne 10. Lors d'une entrée, les caractères sont écrits sur l'écran comme à l'habitude, le curseur avançant chaque fois d'une position vers la droite. Si la colonne 5 est atteinte, alors les touches entrées ne sont tout d'abord pas acceptées et vous voyez au lieu de cela apparaître:

```
Résultat: ...
```

La procédure SAISIE a donc été interrompue. On peut ainsi réécrire la procédure ENTREE1 utilisée précédemment de façon à ce que le masque d'entrée permette d'obtenir plusieurs données par lignes.

```
to ENTREE
MASQUE
make "Nom SAISIE 4 1 9 "
make "Prenom SAISIE 29 1 39 "
make "Rue SAISIE 4 3 19 "
make "Nro SAISIE 26 3 27 "
make "Code SAISIE 12 5 9 "
make "Ville SAISIE 28 5 39 "
op (se :Nom :Prenom :Rue :Nro :Code :Ville)
end
to MASQUE
ct
setcursor [0 1] pr [Nom:...............]
setcursor [22 1] pr [Prenom:...............]
setcursor [0 3] pr [Rue:...............]
setcursor [22 3] pr [Nro:....]
setcursor [0 5] pr [Code postal:.....]
setcursor [22 5] pr [Ville:...............]
end
```

On pourrait naturellement apporter ici diverses améliorations en ce qui concerne le contrôle des valeurs en entrée. Les exemples illustrent cependant déjà comment on peut programmer sous LOGO des masques d'entrée de type professionnel. Ce sujet sera une nouvelle fois abordé à la leçon 14. Nous y présenterons en effet un programme qui permet à l'utilisateur d'écrire le masque directement sur l'écran. Il n'a donc pas besoin d'écrire lui-même le programme MASQUE puisque c'est le programme qui s'en charge. Le LOGO permet en effet de produire des programmes avec d'autres programmes.

Exercices:

1) Dans les premières leçons, les nombres, mots ou listes devaient être écrits directement dans la ligne d'entrée d'appel des procédures. Ces exemples de programmes peuvent maintenant être fournis en valeurs d'entrée avec l'aide de rl, rq et rc. Reprenez donc d'anciens exemples et modifiez les programmes de façon à pouvoir entrer les valeurs nécessaires pendant le déroulement du programme.

2) Des nombres décimaux devront être entrés puis sortis ensuite avec deux chiffres après la virgule. Le nombre devra être précédé d'autant d'espaces que nécessaire pour que le mot de nombre qui en résultera comporte toujours 8 caractères (en comptant les espaces).

3) Formulez une procédure qui permette de commander les mouvements de la tortue sur l'écran en frappant sur certaines touches (par exemple V pour en aVant, R pour en aRrière, D pour rotation sur la droite, etc...).

4) Faites produire des mots au hasard en sélectionnant au hasard des lettres de l'alphabet. Utilisez les mots ascii et char pour cette sélection.

5) Une procédure devra affecter à chaque lettre celle qui suit la suivante. A la fin de l'alphabet, on recommencera au début, Y étant par exemple affecté à A, etc...

6) Développez un programme de code secret (voir l'exercice 5) qui remplacera chaque lettre par une autre de façon irrégulière. Vous pouvez utiliser des nombres aléatoires pour définir ce code si vous veillez, grâce à rerandom, à ce que les séquences parcourues puissent être reproduites.

7) On devra entrer un tableau contenant exactement 10 valeurs par colonne. Les colonnes devront être séparées par un intervalle fixe. Ecrivez un programme qui sorte sur l'écran les nombres entrés, sous forme d'un tableau de ce type.

Leçon 11: Le graphisme avec coordonnées

Nouveaux éléments de vocabulaire:
setpos, setx, $$, seth, towards, sf, setbg, pe, px, setpc, tf, fs, ss, setsplit, setscrunch

Programmes:
CHASSE.TORTUE, SYSTEME.SOLAIRE, BOX, ELLIPSE, POLYGONE, CONSTANTECERCLE, BRIQUET, CAC, SURSAUTER, COURBESINUS, PLOTTER (dessinateur de fonctions avec programmes auxiliaires)

11.1 Le dessin dans un système de coordonnées

Les instructions de base du graphisme avec tortue provoquent des mouvements en avant ou en arrière et des rotations sur la droite ou sur la gauche. Les dessins sont créés sous la forme de traces laissées par la tortue sur l'écran graphique. Ces instructions de base se rapportent à tout moment à l'emplacement et à la direction actuels de la tortue. On peut dire qu'il s'agit d'instructions 'égocentriques'.

La plupart des systèmes graphiques reposent au contraire sur des instructions qui amènent le crayon à dessin dans un emplacement déterminé sur l'écran ou sur le papier. Pour fixer un point, on a besoin d'un système de coordonnées.

Sur le globe terrestre, la situation géographique est par exemple indiquée par la latitude et la longitude d'un lieu. Sur l'écran plat, on utilise les distances les plus courtes d'un point à deux axes de coordonnées, un axe horizontal et un axe vertical. Ces deux axes se coupent dans ce qu'on appelle l'origine des coordonnées.

Si on se demande laquelle des deux méthodes est préférable, la réponse dépendra du type de problème. Si vous voulez amener la tortue dans un emplacement déterminé de l'écran, vous risquez, avec les instructions graphiques étudiées jusqu'ici (fd, bk, rt, lt), de devoir faire de longs tests avant d'arriver exactement au résultat recherché.

Le LOGO permet également de dessiner avec des coordonnées.

Nous savons déjà appeler immédiatement un point bien précis, le centre de l'écran, avec:

> home (amener la tortue dans le centre de l'écran graphique).

Le centre constitue normalement également l'origine des coordonnées pour le LOGO. L'instruction home peut donc être également décrite ainsi: placer le crayon à dessin sur l'origine des coordonnées, c'est-à-dire sur le point de coordonnée (horizontale) x 0 et de coordonnée (verticale) y 0.

De fait, home dirige encore la tortue vers le haut mais nous ne nous intéresserons pas à la direction de la tortue pour le moment.

Pour fixer le crayon à dessin sur un point déterminé, on utilise le mot LOGO setpos. setpos attend comme entrée la liste des deux valeurs de coordonnées, d'abord la coordonnée x puis la coordonnée y.

```
setpos [0 0]
setpos [50 50]
setpos [-100 78]
setpos [-50 -100]
```

Vous voyez probablement maintenant sur l'écran des traits réunissant les points indiqués. Les instructions graphiques sous LOGO doivent être comprises fondamentalement comme des mouvements du crayon à dessin. Le fait qu'un trait soit ou non tracé lors de ces déplacements dépendra du fait que le crayon soit relevé ou abaissé. Si vous faites donc précéder ces instructions de

pu la tortue sera simplement placée successivement sur les points indiqués.

Les coordonnées doivent être fournies sous la forme d'une liste. S'il s'agit de constantes, comme dans les exemples précédents, la liste est simplement produite avec les crochets. Dans tous les autres cas, il faut former une liste avec se:

```
make "x 50 make "y -60 setpos se :x :y
```

Vous pouvez également utiliser d'autres mots que se pour construire des listes.

Quelles valeurs sont autorisées pour les coordonnées?

La réponse dépend de la taille que vous avez sélectionnée pour la fenêtre graphique. La taille de l'écran du CPC AMSTRAD est également très différente de celle du PCW AMSTRAD.

Dimensions standard

Par taille standard nous entendons la taille de la fenêtre graphique dont vous disposez si vous travaillez soit avec l'écran divisé normalement, soit avec la totalité de l'écran disponible. Vous disposez pour cela des instructions

ss (splitscreen) et fs (fullscreen)

On passe également dans l'état splitscreen avec n'importe quelle instruction graphique.

CPC: Coordonnée x -320 à 320,
Coordonnée y -192 (-112) à 191

PCW: Coordonnée x -360 à 360,
Coordonnée y -265 (-94) à 263

Les valeurs minimales entre parenthèses se rapportent ici au réglage standard de l'écran divisé.

L'instruction LOGO

```
setsplit 5
```

permet de fixer la taille de la fenêtre de texte dans l'écran divisé. Vous pouvez vous-même essayer de voir comment se présente alors la limite inférieure pour la coordonnée y en interdisant avec fence un dépassement de la fenêtre graphique et en amenant alors la tortue aussi loin que possible vers le bas jusqu'à ce que le message "Turtle out of bounds" (Tortue en dehors des limites) apparaisse.

Déplacements parallèles aux axes

Avec setpos, il faut indiquer les deux coordonnées. Il existe cependant aussi une forme simplifiée qui permet de n'entrer qu'une des deux coordonnées du point. La seconde coordonnée reste alors inchangée.

setx 100 Fixe sur la valeur 100 la coordonnée x, la coordonnée y ne varie pas.

sety 50 Fixe sur la valeur 50 la coordonnée y, la coordonnée x ne varie pas.

L'entrée n'est pas ici une liste mais un mot au sens de la syntaxe du LOGO. Si l'entrée n'est pas une constante, elle peut être remplacée par toute expression fournissant une valeur numérique:

```
setx 3*:x  ou  sety random 100
```

Si le crayon à dessin est abaissé (pd), setx et sety dessineront toujours des traits parallèles aux deux axes de coordonnées. On peut ainsi dessiner par exemple les axes de coordonnées:

```
to AXES
pu setpos [-320 0] pd setx 320;AXE DES X
pu setpos [0 -112] pd sety 191;AXE DES Y
home
```

```
end
```

Avec les nombres aléatoires, setx et sety permettent aussi d'organiser une chasse à la tortue:

```
to CHASSE.TORTUE
setx (random 320) - 160
sety (random 260) - 129
CHASSE.TORTUE
end
```

Les instructions setpos, setx et sety n'agissent dans un premier temps que sur la position de la tortue; les dessins sont construits par l'intermédiaire des déplacements de la tortue avec le crayon abaissé. DR LOGO dispose cependant également d'un mot permettant de fixer un point déterminé indépendamment de la tortue:

```
dot [100 120]
```

L'entrée pour dot est, comme pour setpos, une liste composée des deux valeurs de coordonnées. Cette liste est formée au moyen des crochets pour les valeurs constantes et au moyen de se pour les noms. Cette instruction correspond à l'instruction de base dans les autres langages de programmation. Nous n'aborderons cependant les applications de dot qu'à partir de la leçon 14.

11.2 La position de la tortue et le système de coordonnées

La tortue peut être amenée sur des points déterminés du système de coordonnées, grâce à setpos, setx, sety. Les deux méthodes graphiques peuvent donc être ainsi combinées. On aura de même besoin dans certains cas de pouvoir déterminer inversement la situation de la tortue dans le système de coordonnées. La fonction tf (abréviation de turtle facts = données sur la tortue) nous fournit toutes les informations nécessaires sur la situation de la tortue.

```
cs setpos [100 120] show tf
```

Réponse: [00 120 0 PD 1 TRUE]

Le résultat fourni par tf est une liste de six éléments. Les deux premiers éléments donnent la position de la tortue dans le système de coordonnées. Nous pouvons également définir un mot POS supplémentaire qui ne sortira que les valeurs des coordonnées.

```
to POS
op piece 1 2 tf
end
```

Le dernier élément de la liste produite par tf indique si la tortue est visible alors que le quatrième indique l'état actuel du crayon à dessin (PD, PU, PE ou PX).

```
ht pu tf
```

Réponse: [100 120 0 PU 1 FALSE]

L'avant-dernier élément indique la couleur du crayon à dessin, ici par exemple le code de couleur 1, et la troisième valeur donne la direction de la tortue mais nous y reviendrons un peu plus tard. Nous allons étudier pour le moment une autre application de la détermination des coordonnées.

Détermination de la constante du cercle

La constante du cercle PI exprime le rapport du périmètre au diamètre. Ce rapport peut être calculé de manière approchée lorsqu'on simule un cercle, par approximation, à partir d'un polygone. Nous espérons que cette méthode ne réveille pas en vous de mauvais souvenirs de cours de maths! Avec le LOGO, cela peut être fait très aisément de façon expérimentale: le périmètre d'un polygone est fourni tout simplement par la somme des côtés et nous pouvons maintenant mesurer le diamètre avec tf.

```
to CONSTANTECERCLE
setx -140
```

```
repeat 180 [fd 2 rt 1]
OP 360*2/((first tf)+140)
end
```

Nous ne dessinons en fait qu'un demi-cercle pour que le diamètre puisse être simplement mesuré comme une modification de la position de la tortue par rapport au point de départ. L'écart par rapport à la valeur exacte de la constante du cercle PI est de 1/50 pour mille seulement!

11.3 La direction de la tortue

La tortue est un crayon à dessin orienté dont la direction est indiquée à l'écran par la pointe du triangle. Lors de l'exécution des instructions:

```
cs et home
```

la tortue est placée au centre de l'écran et dirigée vers le haut. Cette orientation est la position zéro du crayon à dessin. Les indications absolues sur l'orientation de la tortue se réfèrent à cette orientation vers le haut.

La rotation de la tortue s'effectue généralement par rapport à la position actuelle:

```
rt 10 ou lt 10
```

entraîneront une rotation de la tortue de dix degrés. Les angles de rotation peuvent aussi être négatifs. Une rotation sur la droite de -10 équivaut à une rotation sur la gauche de dix degrés. Les angles peuvent aussi être supérieurs à 360 degrés.

Mais il est aussi possible d'indiquer la direction de manière absolue, c'est-à-dire par rapport à la position de départ, comme on indique un point cardinal. On utilise à cet effet le mot LOGO seth.

> seth oriente la tortue vers l'angle indiqué, indépendamment de son orientation précédente.

seth 90 (abréviation de Set Heading)

oriente la tortue vers la droite, c'est-à-dire vers l'Est si on compare à une carte d'état-major. L'entrée pour seth est l'angle par rapport à la position zéro, dans le sens des aiguilles d'une montre.

L'orientation actuelle de la tortue est fournie inversement comme le troisième élément de la sortie de tf:

cs seth 37 pr item 3 tf Réponse: 37

Essayez par exemple:

```
to ARTIFICE
seth random 360
fd random 130
home
ARTIFICE
end
```

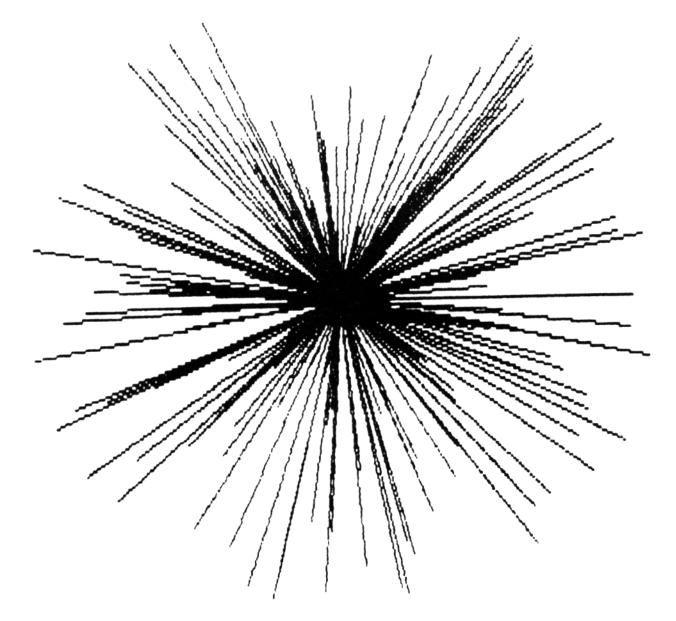


Figure 11.1: Feu d'artifice

La tortue peut également être pointée en direction d'un point déterminé et indiquer la direction correspondante.

towards fournit comme valeur la direction dans laquelle se trouve le point indiqué en entrée par rapport à la position actuelle de la tortue. towards attend en entrée les coordonnées X et Y du point visé. Ces coordonnées doivent être présentées sous forme d'une liste, comme pour setpos.

```
home towards [50 50]
```

Réponse: **45**

Avec la combinaison:

```
seth towards [50 70]
```

la tortue peut être dirigée vers le point de coordonnées (50; 70).

Exemple: construction d'un triangle

La procédure CAC nous servira d'exemple pour illustrer l'orientation absolue.

```
to CAC :c :alpha :b;TRIANGLE COTE ANGLE COTE
(local "bk "gamma)
make "bk tf;Position du point b
seth 90 bk :c lt :alpha fd :b
make "gamma 180 + (item 3 tf) - towards :bk
(pr [Angle Alpha:] :alpha [Beta: ] 180 - :alpha - :gamma
[Gamma: ] :gamma)
setpos :bk
end
```

La procédure dessine un triangle d'après l'indication de deux côtés et de l'angle entre ces deux côtés. Dans l'avant-dernière ligne, on détermine la direction du point de départ grâce à laquelle on peut

alors déterminer également l'angle gamma.

On conserve sous le nom de bk le résultat de tf lorsque la tortue se trouve sur le point B du triangle. On vise ensuite le point B avec towards :bk; ici seule une liste composée des deux valeurs de coordonnées est en fait attendue pour towards. L'expérience révèle que DR LOGO ne s'offusque pas si la liste contient encore d'autres éléments. setpos :bk permet également d'aboutir au but recherché avec la liste de six éléments.

11.4 Les figures dans un réseau de coordonnées

DR LOGO sur les ordinateurs AMSTRAD ne possède pas, dans son vocabulaire de base, de mots propres permettant de dessiner des figures telles que les rectangles, les cercles, les ellipses etc... Examinez les deux exemples suivants:

```
to BOX :l
pu ht setpos :l
pd seth 90
repeat 2 [fd item 3 :l lt 90 fd last :l lt 90]
end

to POLY :co
pu setpos :co pd
POL bf bf :co
end

to POL :co
if emptyp :co [stop]
setpos :co
POL bf bf :co
end
```

BOX dessine un rectangle dont le coin inférieur gauche est défini par les deux premiers éléments et dont la longueur et la largeur sont définies respectivement par les troisième et quatrième éléments de la liste d'entrée. Essayez:

```
fs cs repeat 70 [BOX (se RX RY random 120 random 120)]
```

Pour choisir des points au hasard sur l'écran graphique, on utilisera ici deux procédures auxiliaires RX et RY.

```
to RX
op (random 500) - 300
end
to RY
op (random 300) - 200
end
```

C'est ainsi qu'a été dessinée la figure 11.2.

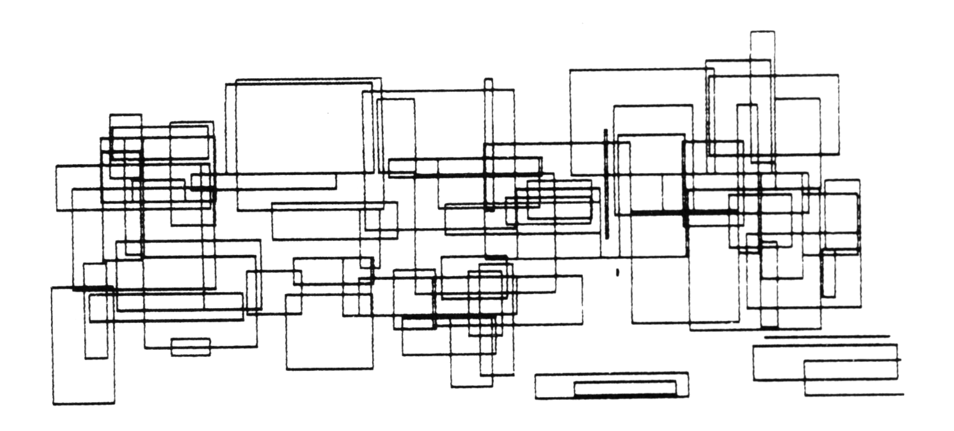


Figure 11.2: Rectangles aléatoires

Pour construire la liste nécessaire pour BOX à partir des indications définissant un rectangle, on utilise se; les crochets ne peuvent être utilisés qu'avec des constantes. Les parenthèses sont nécessaires puisque se est suivi de plus de deux entrées.

La procédure POLY dessine une section de polygone dont les coordonnées des coins sont définies par la liste d'entrée. Le polygone le plus simple est une droite, un triangle constitue un autre exemple:

```
cs
POLY [-100 0 120 150]
POLY [-100 -50 90 -30 0 180 -100 -50]
```

Pour que nous puissions jouer un peu avec POLY, nous définirons tout d'abord une procédure qui produira au hasard les coordonnées d'une section de polygone.

```
to POLYGONE :nombre
local "l make "l []
repeat :nombre [make "l (se :l rx ry)]
op :l
end
```

Nous utilisons ici les fonctions auxiliaires rx et ry qui devront être adaptées chaque fois à la zone souhaitée.

```
fs cs ht POLY POLYGONE 50
```

11.5 Etats de l'écran

La fonction LOGO sf (abréviation de Screen Facts) sort les informations sur l'état de l'écran.

```
sf
```

Réponse: [0 SS 5 WINDOW 1] ou [0 SS 10 WINDOW 0.46875]

La première réponse est la réponse standard sur le CPC AMSTRAD, la seconde celle sur le PCW.

La signification des différents éléments est la suivante:

- Couleur du fond de la fenêtre graphique

- Etat de la fenêtre (TS SS ou FS)
- Division de la fenêtre (Nombre de lignes de texte pour SS)
- Délimitation de la fenêtre graphique (FENCE, WINDOW ou WRAP)
- Rapport des axes

Les différentes valeurs peuvent être modifiées grâce aux instructions suivantes.

- setbg 1 fixe le code couleur pour le fond de la fenêtre graphique.
- ss, fs, ts fixent l'état fenêtre.
- setsplit 7 définit la taille de la fenêtre de texte.
- window, fence, wrap déterminent la réaction en cas de dépassement des limites de la fenêtre graphique.
- setscrunch 0.7 modifie le rapport des axes pour l'amener sur 0.7.

(Les modifications de couleur n'agissent qu'après vidage de la fenêtre avec cs ou clean ou ct.)

Nous allons nous intéresser ici plus particulièrement au rapport des axes. Les écarts entre les différents points image d'un moniteur sont différents selon qu'on a affaire à la direction horizontale ou verticale. Bien entendu, les écarts concrets peuvent également varier d'un moniteur à l'autre. Le LOGO veille cependant à ce qu'on n'ait pas à en tenir compte pour les instructions de dessin. Les différents préréglages sur le CPC ou sur le PCW tiennent compte des différents moniteurs et de leurs différents modes de fonctionnement.

setscrunch permet de modifier volontairement le rapport entre les axes. On aura alors par exemple une procédure CARRE qui dessinera en réalité un parallélogramme, c'est-à-dire un carré étiré ou comprimé. Notez bien qu'une modification du rapport des axes agit également sur la situation d'un point si vous l'appelez par exemple avec setpos. Ce n'est donc pas seulement la forme d'une figure qui sera modifiée mais aussi sa situation sur la fenêtre graphique.

Avec des nombres aléatoires on pourra produire toute sorte d'effets curieux simplement à partir de la procédure CARRE. C'est ainsi que la ligne suivante a permis de réaliser le dessin de la figure 11.3.

```
fs cs repeat 50 [pu setpos se (random 600) - 300 !
(random 200) - 100 pd seth random 360 !
setscrunch 0.1 + (random 20)/10 CARRE 10]
```

L'entrée de CARRE a ici encore une valeur fixe.

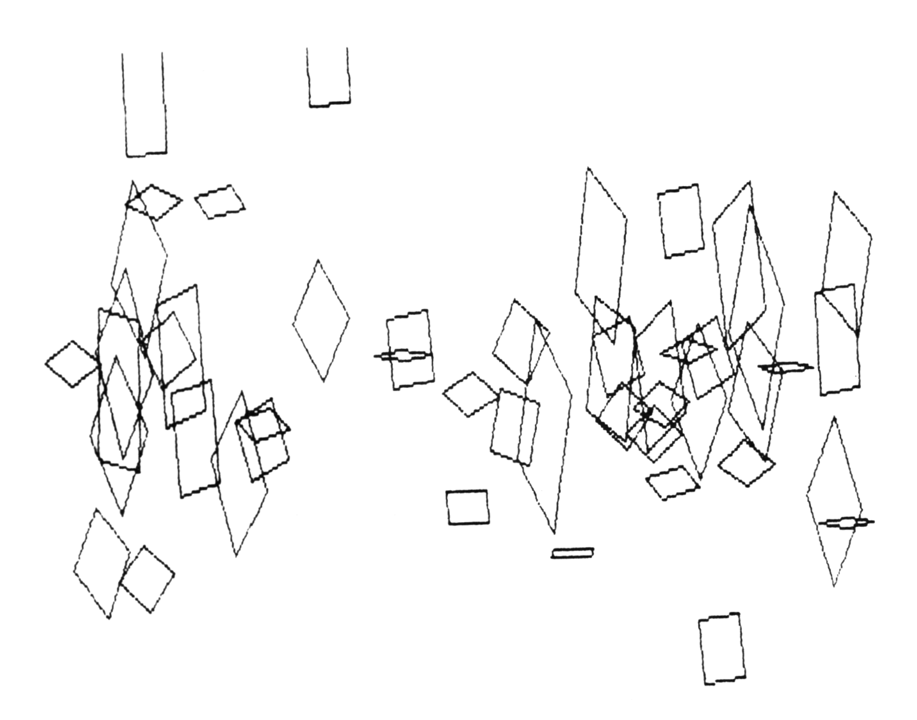


Figure 11.3: **Jeux avec CARRE et setscrunch**

Avec des figures courbes, l'utilisation de setscrunch est encore plus intéressante. C'est ainsi qu'on peut, par étirement ou compression, dessiner des ellipses à partir de cercles alors que ces mêmes ellipses ne pourraient être dessinées sans l'aide de setscrunch que par des calculs très compliqués.

```
to CERCLE :r
pu fd :r rt 95 pd
local "s make "s 0.1745*:r
repeat 36 [fd :s rt 10]
pu lt 95 bk :r
end

to ELLIPSE :a :b
setscrunch :b/:a;sur le PCW, multiplier par 0.46875
CERCLE :a
end
```

Les deux demi-axes de l'ellipse constituent l'entrée pour ELLIPSE, a se rapportant à l'axe horizontal et b à l'axe vertical. On ne peut toutefois utiliser n'importe quelles valeurs pour le rapport entre les axes car DR LOGO n'autorise que les rapport des axes jusqu'à 1:10 ou 10:1. (Cela se traduit un peu différemment sur le PCW du fait du facteur supplémentaire 0.46875).

Pour fêter la visite que nous a faite, fin 1985/début 1986, la célèbre comète de Halley, nous allons prendre comme exemple sa trajectoire. Nous ne prendrons pas en compte les angles entre les niveaux de trajectoire. La procèdure prévoit par ailleurs toutes les planètes du système solaire.

```
to SYSTEME.SOLAIRE :r
ELLIPSE 0.387 * :r 0.379 * :r;MERCURE
ELLIPSE 0.723 * :r 0.723 * :r;VENUS
ELLIPSE :r :r;TERRE
ELLIPSE 1.524 * :r 1.517 * :r;MARS
ELLIPSE 5.203 * :r 5.197 * :r;JUPITER
ELLIPSE 9.539 * :r 9.524 * :r;SATURNE
ELLIPSE 19.18 * :r 19.16 * :r;URANUS
ELLIPSE 30.06 * :r 30.06 * :r;NEPTUNE
```

```
make "k tf seth 0 fd 9.9 * :r
ELLIPSE 39.75 * :r 38.5 *:r;PLUTON
make "k tf seth 90 fd 17.75 * :r seth 0
ELLIPSE 18.34 * :r 4.58 * :r;COMETE DE HALLEY
setpos :k
end
```

La valeur en entrée pour SYSTEME.SOLAIRE est le rayon de la trajectoire de la terre. Suivant votre entrée, tout ou partie des trajectoires entrées seront visibles. Il faut donc faire d'abord un certain nombre d'essais avec la valeur en entrée.

Pour que la comète de Halley soit entièrement dans votre champ de vision, il est préférable de décaler également l'origine des coordonnées. Vous pouvez voir à la figure 11.4 le résultat obtenu.

```
fs cs ht pu setx -140 SYSTEME.SOLAIRE 12
```

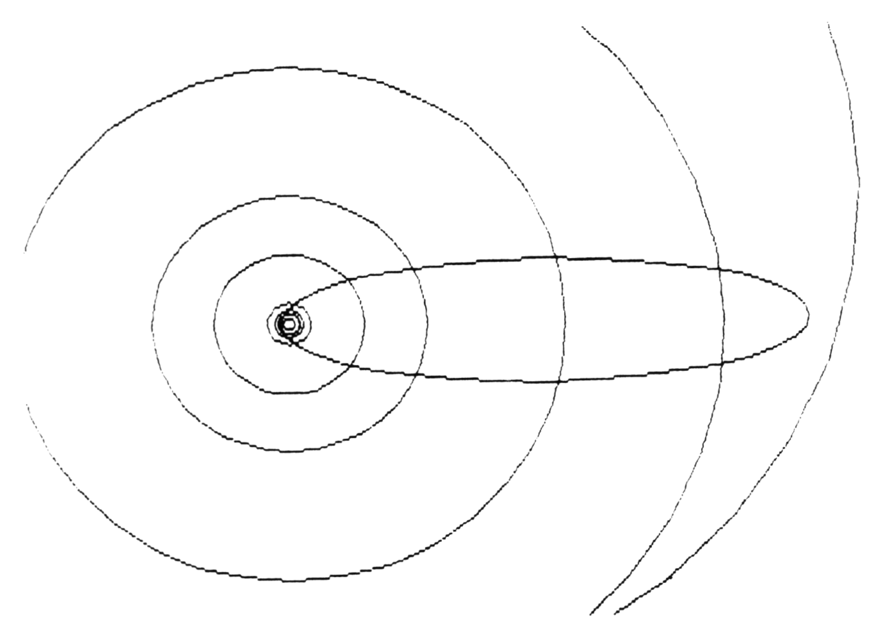


Figure 11.4: La trajectoire de la comète de Halley

11.6 Représentation de fonctions

L'instruction de dessin setpos permet la représentation graphique des valeurs d'une fonction. Pour dessiner la courbe d'une fonction sinus, on peut appeler le programme suivant:

```
to COURBESINUS :x
if :x > 319 [stop]
setpos se :x 100 * sin :x * 2
pd
COURBESINUS :x+6
end
```

Par exemple avec:

```
cs pu COURBESINUS -319
```

La répétition est obtenue ici à nouveau par récursion. Lors de l'appel par récursion, la coordonnée de l'écran est augmentée de 6. Cette précision suffira pour de nombreuses applications et le dessin prend bien sûr plus de temps si on choisit un pas plus petit.

Comme la tortue ne joue pas un rôle important dans le dessin de courbes de fonctions, on peut bien la cacher. Lors de l'appel de COURBESINUS, le crayon à dessin est relevé avec pu. Il est alors amené au début de la courbe, sans qu'apparaisse une trace reliant ce point à la position actuelle de la tortue.

Pour la représentation graphique de fonctions, il faut résoudre quelques problèmes si on veut qu'un programme de dessin produise une image valable pour différents types de fonctions.

Il faut d'abord prévoir une conversion suffisamment souple entre les valeurs absolues sur l'écran et les unités nécessaires pour le calcul des valeurs de fonction. Dans le programme COURBESINUS, x correspond à la coordonnée absolue sur l'écran, la conversion étant effectuée lors de l'appel de la fonction. La valeur de fonction calculée doit être ensuite convertie en valeurs significatives pour l'écran au moyen d'un facteur d'échelle.

Il arrivera parfois aussi que la situation de l'origine des coordonnées, au centre de l'écran, se révèle un obstacle à une utilisation optimale du moniteur. Il faut alors recalculer les coordonnées de façon à obtenir un décalage satisfaisant.

Pour la bonne compréhension d'un graphique de fonction, il est souhaitable de dessiner les axes de coordonnées en marquant les unités sur ces axes.

Un dessinateur de fonction commode

Comparé aux exemples donnés jusqu'ici qui étaient toujours des programmes vraiment courts, le programme de dessin réellement commode que nous allons maintenant vous présenter constitue un projet de programmation de plus grande dimension.

```
to PLOTTER :x0 :y0 :ex :ey :pas;PLOTTER DE FONCTION
fs window
local "bordh make "bordh 190;BORD HAUT
local "bordb make "bordb -190;BORD BAS
local "bordg make "bordg -310;BORD GAUCHE
local "bordd make "bordd 310;BORD DROIT
AXES;DESSINER AXES
MARQUER :ex :ey;MARQUER ECHELLE
pu setpos se :bordg :y0 + FONCTION (:bordg - :x0)/:ex pd
COURBE :bordg - :x0
end
```

C'est le programme principal, celui qui commande les différents composants. Au début, les limites de la fenêtre graphique sont fixées avec les valeurs bordh, bordb, bordg et bordd (les valeurs doivent être modifiées pour le PCW).

```
to AXES;AXES AVEC CADRE
ht cs
pu setpos se :bordg :bordh
pd setx :bordd sety :bordb setx :bordg sety :bordh
pu setpos se :x0 :bordh pd sety :bordh pu
setpos se :bordg :y0 pd setx :bordd
```

```
end
```

AXES dessine les axes ainsi qu'un cadre autour de l'écran.

```
to MARQUER :ex :ey
seth 0 ECHELLEX - :ex * quotient :bordg - :x0 -:ex
seth 90 ECHELLEY - :ey * QUOTIENT :BORDB - :y0 -:ey
end
```

Faites bien attention à écrire correctement les signes moins!

Le programme MARQUER place de petits traits sur les axes, suivant l'intervalle correspondant aux unités ex et ey. Ces marques sont également portées sur le cadre ce qui permet une meilleure orientation.
Les procédures AXES et MARQUER ne nécessitent pas d'entrées. Les valeurs nécessaires sont des noms globaux pour ces deux mots dont la valeur est transmise par le programme d'appel PLOTTER dans lequel ces valeurs sont cette fois des noms locaux. Les programmes auxiliaires pour le programme de dessin ne peuvent être appelés, avec cette formulation, qu'à l'intérieur de PLOTTER.

MARQUER utilise les actions auxiliaires ECHELLEX et ECHELLEY pour fixer les graduations.

```
to ECHELLEX :x;MARQUAGE DANS LA DIRECTION X
if :x > :bordd [stop]
pu setpos se :x :bordh pd fd 8
pu sety :y0 - 5 pd fd 10
pu sety :bordh - 8 pd fd 8
ECHELLEX :x + :ex
end
to ECHELLEY :y;MARQUAGE DANS LA DIRECTION Y
if :y > :bordh [STOP]
pu setpos se :bordg :y pd fd 10
pu setx :x0 - 5 pd fd 10
pu setx :bordd - 8 pd fd 8
ECHELLEY :y + :ey
end
```

Le graphique de la fonction est produit dans la procédure COURBE.

```
to COURBE :x
if :x > :bordd [stop]
setpos se x0 + :x :y0 + :ey * FONCTION :x/:ex
COURBE :x+:pas
end
```

La valeur de la fonction est d'abord calculée au moyen de la procédure FONCTION. Pour le dessin, les valeurs x et y sont ensuite converties en coordonnées écran appropriées. On ne contrôle pas si la courbe dépasse le bord supérieur ou inférieur. Avec l'option window, ce n'est de toute façon pas nécessaire si bordh et bordb correspondent au bord effectif de la fenêtre graphique.

Il nous manque encore la procédure FONCTION qui se charge de calculer la fonction qu'il s'agit de représenter. Le programme de dessin serait encore plus souple si le nom de la fonction n'était pas fixé mais s'il pouvait au contraire être transmis au programme de dessin comme valeur en entrée. Cela est également possible sous LOGO mais nous en parlerons seulement un peu plus tard.

```
to FONCTION :x
op :x * :x
end
```

En l'occurrence, c'est une parabole qui sera ainsi réalisée.

Comme le programme de dessin est souple, vous pouvez tout à fait jouer avec. Pour obtenir une représentation vraiment de qualité, il vous faut rechercher, par des tests répétés, les valeurs idéales pour les unités ainsi que pour la situation de l'origine des coordonnées. Pour la parabole par exemple, il est préférable de décaler l'axe des x un peu plus vers le bord inférieur de l'écran. Il faut choisir les unités de façon à ce que les propriétés intéressantes de la fonction puissent être bien représentées.

```
cs PLOTTER 0 -190 70 20 3
```

Pour la fonction sinus qui est calculée, sous LOGO, en radians, l'unité ne se prête pas au marquage. En effet, un écart significatif sur l'écran doit au moins être long de plusieurs points. La solution peut nous être apportée ici par une graduation appropriée dans le programme de fonction. Pour représenter une autre fonction sinus que celle que nous venons d'utiliser, nous allons ajouter, dans l'exemple suivant, une division par l'argument. Ce graphique est celui de la figure 11.5.

```
to FONCTION :x
op (sin :x * 90)/:x
end

cs PLOTTER 0 0 40 100 3
```

Avec cette modification, l'unité marquée sur les axes représente la valeur 90 degrés. On peut procéder de manière similaire dans d'autres exemples lorsqu'il s'agit de représenter une zone importante sur l'axe des x.

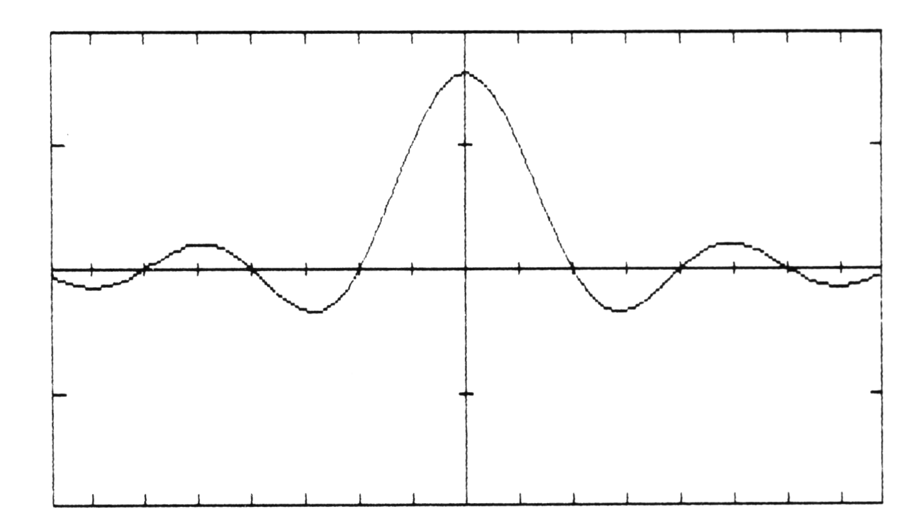


Figure 11.5: Graphique de la fonction sin(x)/x

Qu'est-ce qu'un utilisateur de ce programme doit absolument savoir?

Il doit savoir comment lancer l'action PLOTTER, c'est-à-dire qu'il doit connaître le nom de la procédure ainsi que le nombre et la signification des valeurs attendues en entrée. Dans la version actuelle, il doit aussi savoir que la fonction à représenter doit être définie sous le nom de procédure FONCTION.

Il n'est pas nécessaire que l'utilisateur comprenne le programme lui-même. Il peut tout à fait l'utiliser en effet comme une 'utility', comme un programme utilitaire, c'est-à-dire comme une extension du langage très commode.

Exercices

1) Généralisez le paquet de programmes de dessin de fonctions de façon à ce que plusieurs graphiques de fonctions puissent être représentés en même temps dans un seul dessin.

2) Dessinez le graphique de la fonction sin(1/x). Contournez la valeur x=0 soit par le choix d'un pas approprié, soit par une interrogation spéciale dans le programme FONCTION.

Leçon 12: Les procédures

Nouveaux éléments de vocabulaire:
throw, catch, error, TOPLEVEL

Programmes:
CARREX, HORNER, SQRT, DIVISEUR, POCALL (Analyse de programme avec programmes auxiliaires), STOCKER

12.1 Les actions LOGO en tant que procédures

Vous avez déjà appris plusieurs fois, grâce à quelques exemples, comment programmer en définissant de nouveaux mots LOGO. Les nouveaux mots LOGO désignent des actions exécutées par ce qu'on appelle des procédures. Dans cette leçon, nous examinerons une nouvelle fois la notion de procédure en essayant de l'approcher de façon plus systématique.

Une procédure rend une action LOGO disponible en tant qu'unité. L'action peut se composer à son tour d'une ou plusieurs actions. Dans le cas le plus simple, une procédure ne contiendra qu'une seule activité du système LOGO. Le seul intérêt de la procédure résidera alors dans la définition d'un nouveau nom pour cette action. C'est ainsi que nous pouvons facilement traduire en français le vocabulaire de base du LOGO:

```
to ENAVANT :a
fd :a
end
```

Après cela, le mot LOGO fd pourra être systématiquement remplacé par ENAVANT. On peut de la même manière traduire en français tous les autres mots LOGO.

Cet exemple simple, avec une seule action LOGO, ne permet bien sûr

pas de mesurer toute la signification de la notion de procédure. Le but de la formulation des procédures est, au contraire, de réunir plusieurs opérations sous un nom unique pour faire des actions puissantes à partir d'actions simples. POLYGONE constitue par exemple une illustration de ce que peut être une action encore simple mais néammoins déjà très utile.

```
to POLYGONE :cote :n
repeat :n [fd :cote rt 360/:n]
end
```

Les actions réunies en une procédure doivent former une unité compréhensible. Cela n'a aucun sens d'aligner des pages écran entières d'actions LOGO pour déclarer une seule procédure. Dans les langues naturelles, on n'a pas non plus l'habitude d'employer des mots dont le sens ne puisse être expliqué dans un dictionnaire que par des pages et des pages d'explications.

Les programmes partiels constituant une unité fermée sont aussi appelés sous-programmes dans d'autres langages de programmation. En ce sens, les procédures peuvent aussi être présentées comme des sous-programmes. Mais ce qui caractérise le LOGO, c'est la structure d'ensemble de la programmation qui repose essentiellement sur des actions auto-définies, c'est-à-dire définies par l'utilisateur. Alors que les sous-programmes constituent, dans d'autres langages de programmation évolués, un outil de programmation parmi d'autres, les procédures sont en LOGO le principal instrument de programmation. Il n'existe pas sous LOGO de programme principal à proprement parler. Pour les tâches complexes, il y a tout au plus une procédure-cadre qui réunit sous un même nom la suite des actions partielles.

Cette structure du langage impose qu'un problème soit décomposé en problèmes partiels qui puissent être traités comme des problèmes bien définis et indépendants. Cette méthode d'analyse des problèmes et de conversion des problèmes en programmes informatiques constitue ce qu'on appelle la programmation structurée. Sous LOGO, on ne peut en fait programmer que de façon structurée.

12.2 Les types de procédures

La tâche d'une procédure du type de POLYGONE consiste à effectuer une tâche bien précise, ici le dessin d'un polygone, le résultat étant visible sur l'écran graphique. Cette procédure ne peut cependant pas être exploitée elle-même par d'autres procédures. Il s'agit donc d'une procédure sans sorties.

Lorsqu'un résultat doit être fourni pour être exploité par une autre action, le résultat produit doit être transmis au système LOGO lorsqu'on quitte la procédure. Il s'agit alors d'une procédure avec sortie. Les procédures peuvent aussi être utilisées en même temps pour ces deux types de tâches, la détermination du résultat à sortir ne constituant alors qu'une partie de l'action exécutée par la procédure. C'est là une question qui n'intéresse que la signification intrinsèque de l'action. Pour la syntaxe du LOGO, ce qui importe en premier lieu c'est de savoir s'il y aura une sortie ou non. En effet les procédures avec ou sans sortie sont d'un emploi différent.

Dans le vocabulaire de base du LOGO, il y a des mots qui se terminent par un p. Il s'agit le plus souvent de mots de test tels que memberp, wordp, listp, emptyp, numberp, dont le résultat est soit FALSE, soit TRUE (faux ou vrai).

```
wordp [LOGO]        Réponse: FALSE
wordp "LOGO         Réponse: TRUE
numberp "LOGO       Réponse: FALSE
numberp 3.14        Réponse: TRUE
```

Ces mots LOGO produisent un résultat. Ce sont donc des procédures avec sortie. Le résultat est toutefois très particulier car il ne peut revêtir que la forme d'une des deux valeurs de vérité TRUE/FALSE. Ces mots sont utilisés essentiellement en liaison avec des conditions, c'est-à-dire dans des lignes commençant par if:

```
if numberp :a [pr [a est un nombre !]]
```

De tels mots de test peuvent aussi être déclarés par des procédures. Nous prendrons pour exemple DIVISEURP qui testera si un

nombre naturel est le diviseur d'un second.

```
to DIVISEURP :nombre :diviseur
op remainder :nombre :diviseur = 0
end
```

Cette procédure sortira TRUE ou FALSE suivant que la division nombre/diviseur produira ou non un reste.

Les mots de tests auto-définis tels que celui-ci s'utilisent exactement comme les mots de test faisant partie du vocabulaire de base du LOGO. Les procédures remplissant cette fonction jouent un rôle particulier et constituent ainsi un troisième type de procédures. Nous distinguerons donc trois types de procédures:

- Les procédures sans sortie
- Les procédures avec sortie
- Les mots de test, c'est-à-dire les procédures dont la sortie sera TRUE ou FALSE.

Les procédures avec sortie

Sous LOGO, une procédure ne peut renvoyer qu'un résultat. On l'appelle alors fonction. En ce qui concerne le calcul, le LOGO ne dispose dans son vocabulaire de base que de quelques fonctions mathématiques. Si celles-ci ne suffisent pas, des fonctions spéciales peuvent être programmées en cas de besoin. De nombreuses calculatrices de poche disposent par exemple d'une touche de fonction permettant d'élever un nombre au carré. On aura vite fait d'écrire pour cela une fonction LOGO:

```
to CARREX :x
op :x*:x
end
```

La procédure de fonction suivante vous montre un exemple un peu plus complexe:

```
to HORNER :x :a;SCHEMA DE HORNER
if emptyp :a [op 0]
op (first :a ) + :x * HORNER :x bf :a
end
```

Appel: HORNER 2.4 [2 3 0 5]

Réponse: 78.31999999999

Notez que les parenthèses autour de (first :a) sont nécessaires pour que first soit exécuté en premier et que l'addition ne le soit qu'après. Les opérations de calcul ont donc un rang de priorité supérieur à first, et à d'autres mots LOGO affectant des mots ou des listes, si on n'utilise pas les parenthèses.
La procédure HORNER calcule la valeur d'un polynome d'après ce qu'on appelle le schéma de HORNER. Les coefficients sont transmis sous forme d'une liste, sous le nom a. L'exemple d'appel de HORNER que nous vous avons donné calcule le polynome suivant:

```
2 + 3*X + 0*X^2 + 5*X^3
```

Les polynomes jouent un rôle particulièrement important pour le calcul approché des fonctions mathématiques.

Comme le fabricant n'a pas jugé bon, pour des motifs difficiles à comprendre, d'inclure la fonction de calcul de la racine carrée dans le vocabulaire de base, nous vous présentons maintenant une procédure qui répare cet oubli.

```
to SQRT :x;Racine carrée d'après Heron/Newton
if :x < 0 [pr [Argument negatif] op :x] [op MOYEN (1 + !
:x)/2]
end
to MOYEN :w
local "v make "v (:w + :x /:w)/2
if :w - :v < 1.e-03 [op :v] [op MOYEN :v]
end
```

Procédures à plusieurs résultats

Lorsqu'une procédure produit plusieurs résultats, ceux-ci doivent être réunis sous la forme d'une liste puisque la syntaxe exige qu'il n'y ait qu'un seul objet comme résultat. Comme sous LOGO tout est au fond traité au niveau des listes, plusieurs résultats peuvent toujours être réunis sous forme d'une liste même s'ils sont de contenu et de signification très différents. La limitation à une valeur en sortie est donc une contrainte de pure forme qui ne limite absolument pas les possibilités de la programmation.

Nous allons prendre pour exemple une procédure qui détermine les diviseurs d'un nombre naturel.

```
to DIVISEUR1 :nombre
(local "tl "t)
make "tl [1] make "t 1
repeat (:nombre -2) [make "t :t+1 !
if remainder :nombre :t =0 [make "tl lput :t :tl]]
op lput :nombre :tl
end
```

Appel: DIVISEUR1 30

Réponse: [1 2 3 5 6 10 15 30]

Le programme examine tous les nombres, jusqu'au nombre indiqué, pour savoir s'ils sont un diviseur de ce nombre. Le programme suivant travaille plus vite:

```
to DIVISEUR :nombre
op PARTIE1 :nombre int SQRT :nombre [] []
end
to PARTIE1 :nombre :t :tl :tr
if :t=1 [op (se 1 :tl :tr :nombre)]
if remainder :nombre :t = 0 !
[op PARTIE1 :nombre (:t-1)(se :t :tl)(se :tr :nombre/:t)]
op PARTIE1 :nombre (:t-1) :tl :tr
end
```

Ici, seules les parenthèses de la seconde ligne de PARTIE1 sont nécessaires, les autres n'ont été employées que pour améliorer la présentation. Cette version utilise le fait qu'un diviseur corresponde toujours à un autre plus grand. Vous avez certainement remarqué que PARTIE1 est écrit avec récursion alors que DIVISEUR1 ne contient pas de récursion.

12.3 L'effet combiné des procédures

Une action LOGO peut elle-même mettre en branle d'autres actions LOGO. Cela se produit d'ailleurs déjà dans le programme le plus élémentaire puisqu'il faut bien que des mots LOGO du vocabulaire de base soient utilisés pour qu'on puisse arriver à un effet quelconque. Comme les mots LOGO auto-définis, c'est-à-dire les procédures, sont traités exactement comme les mots du vocabulaire de base, le principe de l'effet combiné des procédures est déjà clairement illustré par des programmes élémentaires.

Si on formule une procédure utilisant le mot LOGO random, il faut savoir:

- quelles entrées la procédure random attend

- quel résultat random fournira.

Peu importe, pour son utilisation dans d'autres programmes, de savoir comment random elle-même travaille. Les liens de connexion pour l'effet combiné de procédures sont donc les entrées et la sortie si sortie il y a. Dans les procédures auto-définies, les noms globaux (variables globales) peuvent éventuellement établir un lien supplémentaire entre les procédures.

Les procédures sans sortie sont utilisées exactement comme les mots LOGO du type fd ou make. De telles actions produisent naturellement un résultat, par exemple certains évènements graphiques, mais elles n'ont pas d'effet en retour sur d'autres procédures. Les actions qui fournissent un nombre comme résultat peuvent remplacer un nombre n'importe où. Là où le nombre 3 pourrait par exemple être employé, on pourrait aussi bien placer SQRT 3 ou HORNER 3 [2 5 7

9]. Lorsque les procédures nécessitent des entrées, celles-ci peuvent aussi être remplacées par l'appel d'une action LOGO qui elle-même fournisse un résultat correspondant à l'entrée attendue.

Par contre, les éléments des listes ne peuvent être remplacés directement par des appels de procédures. Si lors de l'appel de HORNER une liste de coefficients est remplacée par [random 10 SQRT 7], la liste d'entrées se composera de quatre mots: random, 10, SQRT et 7, ce qui n'est pas conforme à ce qui est attendu. La liste d'entrée doit donc être produite avec se:

```
HORNER 2 (se random 10 SQRT 7)
```

Lorsqu'on combine des procédures, il faut naturellement tenir compte du nombre et du type d'entrées nécessaires ainsi que de la sortie éventuelle.

Grandeurs locales et globales

Pour le vocabulaire de base du LOGO, on ne connaît que les exigences concernant les valeurs en entrée et le type du résultat, s'il y a lieu. Peu importe pour l'utilisateur ce qui se passe à l'intérieur du système LOGO lors de l'exécution d'une action de base. D'ailleurs il ne lui est possible d'obtenir aucune information à ce sujet. Toutes les grandeurs ou valeurs qui peuvent apparaître lors de l'exécution de la procédure utilisée n'ont qu'une signification locale, c'est-à-dire qu'elles ne sont connues que de façon interne et seulement sur le moment de l'exécution, à l'exception toutefois des 'variables système' LOGO.

Il est vraiment conseillé d'appliquer le même principe aux mots LOGO auto-programmés. Seules ont alors une signification pour l'action combinées de procédures les valeurs en entrée ou en sortie. Toutes les valeurs qui ne sont ni des valeurs d'entrée ni des valeurs de sortie doivent donc de préférence être définies comme des grandeurs locales de la procédure considérée. C'est à cela que sert l'instruction local.

Les valeurs d'entrée des procédures sont des liens entre celles-ci.

La connexion n'est établie qu'au moment de l'appel d'une action. Pendant le déroulement de la procédure, les valeurs en entrée sont aussi locales. Lors de l'appel d'une action, on transmet pour ainsi dire un paquet dont le devenir ultérieur n'est plus connu qu'à l'intérieur de la procédure qui l'a reçu. On peut donc modifier des valeurs en entrée à l'intérieur d'une procédure, sans que cela entraîne des répercussions extérieures.

Les noms qui ne sont pas des valeurs d'entrée et qui n'ont pas non plus été déclarés avec local comme des noms locaux, sont connus de toutes les procédures en tant que noms globaux. pons vous permet de savoir quels noms sont de ce type et quel est leur contenu. Du fait de leur signification propre, il peut être très intéressant d'attribuer certaines valeurs à des noms globaux lorsqu'il apparaît intéressant de disposer de ces valeurs, comme par exemple le constante du cercle PI, pour toutes les procédures possibles.

En utilisant des noms globaux, on peut éviter d'avoir à transmettre les valeurs nécessaires comme valeurs d'entrée d'autres procédures. Pour que cela n'entraîne pas de conflits, il faut se souvenir de tous les noms globaux déjà utilisés lors de chaque utilisation ultérieure de mots LOGO.

Les noms globaux peuvent être utilisés également pour la sortie. Au lieu de:

```
op 1 + random 6
```

on peut utiliser également la ligne:

```
make "A 1 + random 6
```

si A n'avait pas été déclaré comme nom local. Le résultat pourra être employé comme valeur de la variable A plutôt que comme résultat de la procédure correspondante.

L'emploi de noms globaux peut permettre d'obtenir des programmes plus courts, notamment lorsqu'on a affaire à des procédures plusieurs fois imbriquées les unes dans les autres. Dans ce cas, il faut mettre en balance d'un côté les avantages de la technique

modulaire de programmation et d'un autre côté l'intérêt que peuvent représenter des textes de programme plus courts.

12.4 Examen de la structure des programmes

Sous LOGO, de petits composants travaillent ensemble de manière à remplir ainsi des tâches plus complexes. Cela peut se dérouler à plusieurs niveaux, de sorte par exemple qu'une procédure peut en déclencher une autre qui à son tour pourra appeler d'autres actions LOGO. Une procédure LOGO peut aussi s'appeler elle-même, soit directement, soit à travers des étapes intermédiaires. C'est en fait cette action combinée de plusieurs procédures qui représente la structure logique d'un programme d'envergure écrit en LOGO.

Un programme LOGO déclenche avec des mots LOGO des actions et il importe peu au fond que ces mots soient des mots définis par l'utilisateur ou des mots faisant partie du vocabulaire de base. Pour l'emploi des procédures définies par l'utilisateur, la seule chose qui compte, outre leur fonction intrinsèque, est avant tout le nombre et la signification des entrées qu'elles attendent. Il n'est pas absolument indispensable de savoir quelles procédures seront à leur tour mises en jeu indirectement à la suite de cet appel.

Pendant la phase de développement d'un programme, il peut néammoins être utile pour un programmeur de pouvoir suivre en détail le déroulement logique du programme sur tous les différents niveaux. Certaines versions plus complètes du LOGO offrent déjà dans leur vocabulaire de base des mots spécifiques pour une telle analyse du déroulement d'un programme. Nous allons donc vous présenter dans cette section un petit paquet de programmes que vous pourrez ensuite charger et employer lorsque vous en aurez besoin.

Et, de fait, il est parfaitement possible d'analyser des programmes LOGO à l'aide d'autres programmes LOGO. Nous utiliserons pour cela certains éléments de langage que nous ne décrirons de façon détaillée que dans les chapitres ultérieurs. Nous vous demandons donc ici dans un premier temps de jouer le rôle d'un utilisateur qui ne s'intéresse qu'au fonctionnement concret du programme.

Les programmes présentés auront pour fonction de réaliser le mot LOGO

```
POCALL
```

(Ce terme est par ailleurs emprunté au mot du vocabulaire de base de la version DR LOGO pour l'ATARI ST qui permet d'obtenir cette fonction.) Le mot POCALL peut d'ailleurs être également appliqué à lui-même, examinons par conséquent tout simplement le résultat obtenu avec

```
POCALL "POCALL
```

```
0 : POCALL 1: ...$ANALYSE
2 : ......$LISTE
3 : .........$PHRASEL
2 : ......$ANALYSE*
```

La procédure POCALL nous a ainsi révélé quelles procédures sont utilisées lorsqu'elle est appelée. A l'intérieur de POCALL même, c'est-à-dire au niveau 1, c'est la procédure $ANALYSE qui est appelée. On nous indique pour le niveau 2 un appel de $LISTE, ce qui signifie que $LISTE est appelée par $ANALYSE, $LISTE appelle alors à son tour le programme $PHRASEL et c'est pourquoi ce dernier appel se déroule au niveau 3. On nous indique ensuite à nouveau un appel au niveau 2, c'est-à-dire que cet appel est effectué à partir de la procédure $ANALYSE. C'est d'ailleurs le programme $ANALYSE lui-même qui est alors appelé et il s'agit donc d'un appel récursif.

Lors d'un auto-appel ou aussi lors d'appels récursifs mutuels de procédures, il devient tout à fait sans intérêt de continuer à suivre l'analyse des appels réciproques car cela ne pourrait que déboucher sur une répétition sans fin et ne nous donnerait de toute façon pas d'informations supplémentaires. C'est pourquoi l'appel récursif est marqué par une étoile.

L'étoile ajoutée au nom de la procédure permet ainsi de découvrir facilement où des récursions sont utilisées dans un programme

relativement complexe.

Voici le paquet de programmes pour l'analyse de la structure des programmes:

```
to POCALL :p
;ern glist ".APV (local "$funk "$li
make "$funk glist ".DEF
make "$li []
$ANALYSE :p 0
end

to $ANALYSE :$ :n
(type :n ": ) repeat :n [type "...] type :$
if memberp :$ :$li [pr "* stop] [pr "]
if not namep :$ [$LISTE :$]
(local "ta "l)
make "ta thing :$
make "l 1 make "$li lput :$ :$li
if emptyp :ta [make "$li bl :$li stop]
repeat count :ta $ANALYSE item :l :ta :n+1 make "l :l+1]
make "$li bl :$li
end
```

Vous avez certainement remarqué que dans le paquet de programmes présenté plusieurs noms commencent par le caractère $. Le but essentiel de l'emploi de ce caractère est d'éviter des coïncidences dûes au hasard avec les noms employés dans les programmes à analyser.

Au début de la procédure POCALL, les noms $funk et $li sont définis pour des listes. Avec la ligne

```
make "$funk glist ".DEF
```

(notez bien que DEF est écrit en majuscules) tous les noms de procédure définis au moment de l'appel sont réunis sous le nom $funk. La liste $funk constitue le réservoir contenant toutes les procédures possibles.

Les noms identifiés les uns après les autres au cours de l'analyse comme des procédures appelées sont conservés dans la liste $li qui est dans un premier temps créée dans POCALL seulement comme une liste vide. C'est au vu de cette liste que les appels récursifs pourront ensuite être identifiés.

Le programme $ANALYSE effectue l'examen véritable de la procédure entrée comme valeur :p, la seconde entrée indiquant le niveau de l'appel de procédure. Chaque fois que $ANALYSE est appelé, un appel de procédure a été identifié qui immédiatement annoncé. Si le nom de procédure en question est déjà contenu dans la liste $li, il s'agit d'un appel récursif qui entraîne la sortie du signe * et la fin de l'analyse.

A la quatrième ligne, le test

```
namep :$
```

contrôle si le nom de procédure entré comme nom est lié à une valeur. Le mot LOGO répond TRUE si le mot entré désigne une variable déjà définie. Si vous entrez par exemple

```
pr :inconnu,
```

le LOGO vous répondra par le message d'erreur "inconnu has no value". Le fait que ce message doive apparaître peut déjà être décelé à la réponse FALSE lors de l'emploi du mot de test namep

```
namep "inconnu
```

Réponse: FALSE

A quoi sert donc l'interrogation de :$ avec namep? Lorsque l'appel d'une procédure a été identifié, la procédure en question doit elle-même être examinée. C'est à cela que sert la procédure $LISTE car $LISTE produit pour une procédure donnée une liste qui contient elle-même à son tour les noms des procédures qui y sont appelées. La liste des noms de procédures rencontrés est ici affectée comme valeur à un nom qui correspondra au nom de la procédure. Essayez:

```
$LISTE "$ANALYSE pr :$ANALYSE
```

Réponse: [$LISTE $ANALYSE]

Notez que le nom $ANALYSE est maintenant aussi bien un nom de procédure que le nom d'une variable qui contient justement les noms de programmes appelés dans la procédure de même nom. Vous voyez donc que l'emploi simultané d'un même nom aussi bien pour une procédure que pour une variable est parfaitement autorisé en LOGO. Pour éviter tout risque de confusion, il est cependant recommandé de n'user de cette possibilité qu'avec prudence. L'identité des noms a aussi pour conséquence de faire que vous trouviez maintenant lors de l'appel de l'éditeur la ligne

```
make "$ANALYSE [$LISTE $ANALYSE]
```

à la suite du texte du programme.

L'interrogation namep :$ permet donc de déterminer si on sait déjà quelles procédures sont appelées dans le programme concerné. Si ce n'est pas le cas, c'est-à-dire si la procédure dont il s'agit n'est pas encore apparue au cours de l'analyse, cette liste sera précisément produite par l'appel de $LISTE.

Cette liste est maintenant affectée provisoirement au nom ta dans la sixième ligne de $ANALYSE car la liste sera utilisée à plusieurs reprises. On utilise pour cela le mot LOGO

```
thing
```

Si notre ligne se présentait en effet ainsi:

```
make "ta :$
```

c'est alors le nom de la liste, $ANALYSE en l'occurrence, et non la liste voulue elle-même, qui serait disponible sous ta. On pourrait imaginer de placer pour cela un deuxième double point devant le nom concerné mais ce ne serait pas conforme à la syntaxe du LOGO.

```
thing :$
```

signifie donc:

```
valeur de :$
```

La septième ligne range enfin avec lput le nom de programme qui vient d'être examiné dans la liste des procédures rencontrées jusqu'ici.

Les programmes ne peuvent cependant pas appeler d'autres programmes sur un nombre de niveaux illimité. Si une procédure n'appelle elle-même aucune autre procédure, $LISTE produit une liste vide. Dans ce cas l'appel récursif de $ANALYSE à la huitième ligne prendra fin. Si une procédure en appelle par contre d'autres, le programme $ANALYSE doit être appelé par récursion dans la neuvième ligne pour chaque appel de procédure.

Une fois cette analyse terminée, la procédure examinée est à nouveau retirée, dans la dernière ligne, de la liste $li.

```
to $LISTE :$T
(local "p "l)
make "p $PHRASEL text :$t
make "l 1 make "$t []
repeat count :p [make "a item :l :p if memberp :a !
:$funk [make :$t se thing :$t :a] make "l :l+1]
end

to $PHRASEL :$
local "l$ make "l$ []
label "ANF if emptyp :$ [op :l$]
if listp first :$ [make "$ se first :$ bf :$ go "ANF]
make "l$ se :l$ first :$
make "$ bf :$ go "ANF
end
```

La fonction de la procédure $LISTE, qui utilise $PHRASEL comme programme auxiliaire, a déjà été évoquée.

Dans la troisième ligne, le texte complet de la procédure est dans un premier temps produit sous forme de liste avec

```
text :$t
```

et cette liste est transmise au programme auxiliaire $PHRASEL. Le mot LOGO text sera expliqué plus en détail ultérieurement. Le texte du programme devient ainsi disponible sous le nom d'une variable et il pourra de ce fait être lui-même traité par des programmes. L'appel d'exemple POCALL "POCALL montre bien que les procédures peuvent même traiter leur propre texte de programme.

Essayez un peu

```
text "POCALL
```

Le texte du programme est renvoyé sous forme d'une liste dont les différents éléments représentent chaque fois une ligne de programme sous forme de liste. Le premier élément contient les noms des paramètres d'entrée. Comme les mots LOGO comme repeat et if apparaissent également dans des listes, les différentes lignes de programme peuvent être à leur tour elles-mêmes des listes plusieurs fois imbriquées.

Cette forme se prête mal à la recherche de noms de procédure dans le texte du programme car le mot LOGO memberp ne peut être appliqué avec l'effet voulu qu'aux phrases. C'est pourquoi la procédure $PHRASEL convertit tout d'abord le texte du programme en forme de phrase, c'est-à-dire d'une liste de mots. La procédure élimine les imbrications dans les listes et elle peut aussi être appliquée à cet effet à d'autres listes.

```
$PHRASEL [Ce [fut [une [liste]] imbriquee]]
```

Résultat [Ce fut une liste imbriquée]

L'élimination de l'imbrication des listes est effectuée dans $PHRASEL avec les deux mots LOGO

```
listp et se
```

Le mot de test listp détermine ici si l'entrée est une liste. La liste est décomposée à l'aide de first et de bf puis elle est

ensuite reconstituée avec se, les crochets extérieurs étant ainsi chaque fois éliminés. Si vous voulez suivre le déroulement de cette opération dans le détail, vous pouvez activer trace avant l'appel de la procédure.

Remarques de conclusion

Dans l'exemple de l'auto-analyse POCALL "POCALL, les procédures appelées dans le programme $ANALYSE sont placées dans la variable globale $ANALYSE. Si POCALL est utilisé plusieurs fois, cette liste ne sera pas produite à nouveau si elle existe déjà. Avec des modifications de programme ultérieures, il peut en principe arriver que les appels de procédures disparaissent ou viennent s'ajouter mais d'autres appels de POCALL ne permettront pas de le remarquer.

Si une telle situation se produit, le mieux est d'effacer tous les noms globaux avec la ligne

```
ern glist ".APV
```

avant d'appeler POCALL. C'est pourquoi cette ligne figurait sous forme d'une ligne de commentaire au début de la procédure POCALL. Mais comme cela peut également entraîner l'effacement indésirable d'autres noms, nous avons rendu cette ligne inopérante en la faisant commencer par un point-virgule.

Révélons pour terminer que l'on peut, avec des méthodes que nous aurons à discuter plus tard, construire des noms de programmes pendant le déroulement d'un programme et qu'on peut même définir de nouvelles procédures par programme. Dans des applications aussi sophistiquées, il n'est absolument pas possible de déterminer d'avance, au vu du texte du programme, tous les appels de programmes qui se produiront effectivement.

12.5 Sauts sur plusieurs niveaux

En règle générale, les procédures sont abandonnées avec stop, op ou lorsqu'est atteinte la fin de la procédure. Le système LOGO revient

ainsi au programme d'appel. Parfois, par exemple pour des entrées spéciales, on souhaitera revenir en mode direct sans que soient exécutées d'autres actions. On peut utiliser pour cela throw "TOPLEVEL. On évite ainsi de devoir prévoir dans toutes les procédures d'appel des mesures pour un cas particulier.

Lors du retour au mode direct, le système LOGO doit effectuer lui-même les tâches nécessaires pour terminer les procédures parcourues. C'est pourquoi il serait également souhaitable de pouvoir revenir non pas totalement au mode direct mais seulement de quelques niveaux en arrière.

Un saut à l'intérieur d'une procédure peut être exécuté avec go. Le retour à une procédure d'appel par dessus plusieurs niveaux de l'appel de procédure est réalisé par throw. De même que GO, throw a besoin d'un but de saut. Sans autre précision, le LOGO connaît le but de saut TOPLEVEL que nous avons déjà utilisé. Pour définir vous-même des buts de saut pour throw, vous devez employer le mot LOGO catch qui va de pair avec throw comme label avec go.

```
to DEMO
catch "BUT [NIVEAU2]
pr [BUT DE SAUT DANS DEMO ATTEINT]
end

to NIVEAU2
pr [NIVEAU2 ATTEINT]
NIVEAU3 pr [DANS NIVEAU2, RETOUR DE NIVEAU3]
end

to NIVEAU3
pr [MAINTENANT AU NIVEAU3]
pr [DOIS-JE REVENIR A DEMO?]
if lc first rq ="o [throw "BUT]
pr [DOIS-JE INTERROMPRE TOUT DE SUITE?]
if lc first rq ="o [throw "TOPLEVEL]
pr [JE CONTINUE!]
end
```

Cet exemple a seulement pour but d'illustrer throw et catch. Si

vous répondez Oui à la question dans NIVEAU3, vous voyez ensuite apparaître le message "But du saut atteint". Si votre réponse n'est pas Oui, NIVEAU2 vous annonce le retour du NIVEAU3. Dans ce cas également, le but du saut est atteint après que NIVEAU2 ait été terminé car c'est la prochaine instruction après l'appel de NIVEAU2.

Le but du saut TOPLEVEL (Attention aux majuscules!) est défini à l'intérieur du système LOGO. Le LOGO revient alors au mode direct sans retourner dans les procédures d'appel comme ce serait le cas avec stop. Le throw "TOPLEVEL agit donc un peu comme une interruption avec la touche stop sans toutefois que cela entraîne l'affichage d'un message spécial sur l'écran.

catch a deux entrées. La première est un mot désignant le but du saut alors que la seconde est une liste contenant des instructions exécutables.

Notez bien que: le saut au but désigné dans l'instruction catch n'est possible qu'à partir de procédures qui ont été appelées par les procédures énumérées dans la liste d'entrée! Le but doit donc, en d'autres termes, être trouvé dans une procédure qui sera parcourue lors du retour au mode direct. Par rapport à cette piste, le nom du but du saut est local, c'est-à-dire qu'il n'est pas connu à l'extérieur. C'est pourquoi il peut être utilisé plusieurs fois dans différentes pistes sans conflit.

12.6 Traitement programmé des erreurs

En cas d'erreur, un message d'erreur apparaît normalement dans l'écran de texte puis le LOGO revient en mode direct. Vous pouvez alors parvenir immédiatement dans l'éditeur en entrant

```
ed
```

et le curseur se trouvera le plus souvent sur la source de l'erreur.

Cette façon de réagir aux erreurs est en général très pratique. Le

LOGO est un langage typique avec interpréteur. Si une erreur se produit, le programme erronné peut être corrigé et l'exécution peut être relancée du début. Les valeurs incorrectes qui provoquent une erreur peuvent elles aussi être modifiées avant une nouvelle tentative. Les noms globaux peuvent même être traités avec l'éditeur ce qui pour des données un peu compliquées est beaucoup plus simple que de les fixer à nouveau avec make.

Mais il en va tout autrement pour l'utilisateur d'un programme LOGO qui ne maîtrise pas lui-même ce langage. Les bons programmes d'application ne doivent bien sûr pas contenir d'erreurs mais il faut aussi qu'ils puissent réagir à des entrées incorrectes de l'utilisateur. C'est pourquoi il serait intéressant de pouvoir utiliser les possibilités existantes pour intercepter une erreur.

Lorsqu'une erreur se produit, le système LOGO essaie automatiquement d'exécuter l'instruction throw "error, c'est-à-dire un retour à un catch "error. Si ce but de saut a effectivement été prévu dans le programme, le saut est exécuté et aucun message d'erreur n'est sorti. catch "error doit se trouver dans une procédure qui sera parcourue lors du retour au mode direct, de même que pour une instruction throw normale. catch "error peut cependant être prévu simultanément dans différentes pistes.

Dans la plupart des applications de qualité, la source d'erreur est évidente dès le stade de la programmation. On peut cependant également obtenir sous forme de liste le message correspondant à la dernière erreur apparue à l'aide de la fonction error. Essayez simplement:

```
show error
```

La ligne suivant le catch "error peut également être atteinte après que les instructions énumérées dans la liste d'instructions aient été régulièrement terminées. En cas de besoin, on peut déterminer le cas présent grâce à un test.

```
if emptyp error [...]
```

S'il n'y a pas eu d'erreur, on obtiendra ici la valeur de vérité

TRUE.

La procédure STOCKER représente une application simple. Lors de la sauvegarde sur disquette avec save, les fichiers existant déjà sur la disquette ne peuvent être effacés. Si le nom de fichier utilisé avec save existe déjà, vous obtenez un message d'erreur qu'il nous est possible d'intercepter avec catch "error.

```
to STOCKER :fichier
catch "error [save :fichier]
if emptyp error [stop]
pr [Fichier] :fichier [existe déjà!])
pr [Supprimer ancienne version? o/n]
if lc rc ="o [erasefile :fichier save :fichier] !
[pr [Interrompu sans sauvegarde!]]
end
```

Si le nom de fichier existe déjà, vous obtenez normalement un message d'erreur lorsque vous tentez une sauvegarde avec save. Vous pouvez naturellement réagir à ce message d'erreur en mode direct. La procédure STOCKER nous permet toutefois des réactions contrôlées par programme. Si l'utilisateur le souhaite, l'ancienne version du fichier sera supprimée et le contenu de la mémoire de travail sera sauvegardé sous le même nom.

Les bons système de programmation procèdent souvent de la façon suivante: le fichier déjà existant sur la disquette est conservé sous un autre nom et en cas d'erreur son contenu reste donc disponible. C'est théoriquement possible avec DR LOGO puisque vous trouverez dans le manuel la description du mot LOGO

```
changef
```

pour renommer des fichiers. Malheureusement cette instruction ne fonctionne pas sur les systèmes LOGO actuellement disponibles pour les ordinateurs AMSTRAD: vous voyez toujours apparaître le message d'erreur File ... not found.

Exercices

1) Utilisez POCALL avec des programmes des leçons précédentes.

2) Formulez un mot de test VOYELLEP qui sortira TRUE ou FALSE comme résultat suivant que le caractère entré sera ou ne sera pas une voyelle.

3) Une procédure devra rechercher les voyelles faisant partie du mot entré et les sortir sous forme d'une liste.

4) L'entrée devra être une liste de nombres. On calculera la somme de ces nombres qui sera renvoyée comme résultat.

Leçon 13: La récursion

Nouveaux éléments de vocabulaire:
Récursion avec auto-appel à la fin, déroulement de la récursion avec auto-appel en milieu de procédure

Programmes:
CLASSEMENTS (Probabilités), TRUCS (coefficient de binôme), PGDC, PPMC (algorithme d'Euclide), TETRA (récursion graphique), ARBRE (récursion graphique), TOUR (Tour de Hanoï avec procédures auxiliaires), STRATEGIE (version itérative des tours de Hanoï)

13.1 Appel récursif à la fin de la procédure

Les programmes avec récursion sont caractéristiques du LOGO. C'est pourquoi nous les avons d'ailleurs présentés assez tôt. Dans cette leçon, nous allons nous livrer à une étude systématique de la récursion.

Dans le cas de figure le plus simple, l'auto-appel ne sert qu'à une répétition sans aucune modification. Une étape graphique de base telle que le dessin d'un angle peut être ainsi répétée autant de fois qu'on le veut. Comparez maintenant deux programmes qui effectuent cela avec le même résultat:

```
to ANGLE1
fd 60 rt 100
ANGLE1
end

to ANGLE2
label "debut fd 60 rt 100 go "debut
end
```

La répétition de l'étape de base constituée par un mouvement en

avant et une rotation sur la droite est atteinte dans ANGLE1 par l'auto-appel et dans ANGLE2 par l'instruction de saut go. ANGLE1 est une formulation récursive et ANGLE2 une formulation itérative de la même tâche.

Lors de l'auto-appel, les valeurs en entrée peuvent être modifiées. La modification de valeurs lors de répétitions peut être formulée de façon particulièrement simple grâce à la récursion:

```
to ANGLE3 :angle :longueur
fd :longueur rt :angle
ANGLE3 :angle :longueur+1
end
to ANGLE4 :angle :longueur
lable "debut fd :longueur rt :angle
make "longueur :longueur + 1
go "debut
end
```

Le premier programme est à nouveau une solution récursive et le second une solution itérative de la même tâche. On peut faire apparaître la différence dans l'exécution avec le mode trace ou le mode watch du LOGO. Avec la formulation itérative ANGLE4, le mode trace n'indiquera que la modification de LONGUEUR par make alors que pour la variante récursive ANGLE3 l'auto-appel répété sera indiqué avec les valeurs d'entrée actuelles.

Il est aisé de voir ce qui doit se produire lors d'une récursion: à la fin de ANGLE3, il s'agit de dessiner un nouvel angle dont la longueur devra être supérieure de un. Avec la répétition itérative dans ANGLE4, il faut d'abord comprendre comment se déroule la répétition avant de pouvoir en comprendre l'objet. Cela ne pose bien sûr aucun problème pour un exemple aussi simple mais pour des problèmes plus complexes, il peut très vite devenir difficile de comprendre la signification concrète des répétitions.

Avec la répétition avec make, il faut comprendre qu'il s'agit ici de retirer le contenu d'un tiroir (longueur), de calculer un résultat avec ce contenu puis de replacer ce résultat dans le même tiroir.

Avec la récursion, il peut d'un autre côté devenir difficile de comprendre le déroulement interne d'un programme. Que se passe-t-il dans un ordinateur lorsqu'un programme s'appelle lui-même? N'est-ce pas comme un chien se mordant la queue? Voilà les questions qui se posent à un utilisateur du LOGO qui a une longue expérience de travail sur des langages de programmation sans récursion. Le LOGO est conçu de telle façon que le programmeur n'ait pas finalement à s'occuper des opérations se déroulant à l'intérieur de l'ordinateur. Dans le programme LOGO est formulé ce qui doit se passer concrètement, le système LOGO s'occupe du reste. Pour la formulation des objectifs concrets, les solutions récursives sont plus directement axées sur la définition du problème alors que les solutions itératives sont généralement mieux adaptées au traitement interne dans l'ordinateur. C'est pourquoi les solutions itératives seront souvent plus rapides d'exécution que les solutions récursives.

La récursion simple, dont il sera question dans cette section et qui se définit par le fait que l'appel récursif constitue la dernière instruction de la procédure et qu'aucun résultat n'est transmis à la procédure, est cependant exécutée relativement rapidement sous LOGO. Elle peut être parcourue autant de fois que souhaité. En mode watch ou trace, DR LOGO montrera toujours le niveau le plus bas [1].

Si la procédure a des entrées, que celles-ci soient modifiées ou non dans la récursion, la récursion pose déjà plus de problèmes. En mode watch ou trace, un niveau plus élevé sera chaque fois indiqué lors de l'appel récursif ([2], [3], ...). Une comparaison révèle que la récursion n'est pas réalisée dans la version dont vous disposez de façon aussi puissante que sur d'autres versions du LOGO pour des ordinateurs comparables.

13.2 Les appels récursifs avant la fin de la procédure

Dans la récursion qui ne sert qu'à la répétition, l'auto-appel constitue la dernière action à l'intérieur de la procédure. Par contre, s'il y a ensuite encore d'autres lignes de programme, le déroulement du programme lors de la récursion sera plus difficile à

suivre. Comment une action peut-elle se mettre en oeuvre elle-même alors qu'elle n'a pas encore été achevée?

Examinons un exemple simple dans lequel on se contente de compter des répétitions:

```
to COMPTER1 :nombre
if :nombre > 5 [stop]
pr :nombre
COMPTER1 :nombre + 1
end

to COMPTER2 :nombre
if :nombre > 5 [stop]
COMPTER2 :nombre + 1
pr :nombre
end
```

Dans ces deux programmes, il s'agit de compter jusqu'à 5. Pour que la répétition s'interrompe, une condition d'interruption a été prévue dans les deux exemples grâce à if ... [stop]. Avec COMPTER1, l'auto-appel constitue à nouveau la dernière instruction de la procédure alors que dans COMPTER2 il y a encore après cela l'instruction d'impression.
Après l'appel de:

```
COMPTER1 1
```

la valeur initiale est augmentée et le résultat 1, 2, 3, 4, 5 est sorti. Sans la ligne de programme avec if, les nombres entiers seraient tous sortis dans l'ordre jusqu'à une intervention manuelle extérieure. Après l'appel de:

```
COMPTER2 1
```

nous voyons par contre 5, 4, 3, 2, 1 apparaître sur l'écran. La cause de cet ordre inversé tient au changement de position de l'instruction de sortie pr :nombre.

Que se passe-t-il si la condition d'interruption avec if est ôtée

de la procédure COMPTER2? Le plus simple pour soustraire temporairement une ligne de l'exécution du programme consiste à la transformer en commentaire. Placez donc un point-virgule au début de cette ligne. Vous ne verrez alors rien au début après avoir appelé COMPTER2. Mais ensuite, le LOGO annoncera une interruption avec:

```
Out of LOGO stack space
```

(Mémoire de pile du LOGO épuisée)

Un tel appel récursif ne peut donc pas être répété aussi souvent qu'on le veut. Le système LOGO a visiblement besoin ici de place mémoire, de ce qu'on appelle une pile car la zone prévue est limitée.

Même pour COMPTER1 la pile est nécessaire et la répétition ne peut se produire indéfiniment. Il s'agit cependant dans ce cas d'un défaut tout à fait superflu de la version du LOGO pour les ordinateurs AMSTRAD car il est inutile de conserver quoi que ce soit pour plus tard lorsque l'appel récursif se produit à la fin de la procédure.

La différence entre le déroulement de COMPTER2 et celui de COMPTER1 peut ici encore être examinée avec trace ou watch. Nous avons préféré watch en l'occurrence car c'est plus court.

```
watch
COMPTER1 3
[1] In COMPTER1, if :nombre > 5 [stop]
[1] In COMPTER1, pr :nombre - Réponse: 3
[1] In COMPTER1, COMPTER1 :nombre+1
[2] In COMPTER1, if :nombre > 5 [stop]
[2] In COMPTER1, pr :nombre - Réponse: 4
[2] In COMPTER1, COMPTER1 :nombre+1
[3] In COMPTER1, if :nombre > 5 [stop]
[3] In COMPTER1, pr :nombre - Réponse: 5
[3] In COMPTER1, COMPTER1 :nombre+1
[4] In COMPTER1, if :nombre > 5 [stop]
```

```
COMPTER2 3
[1] In COMPTER2, if :nombre > 5 [stop]
[1] In COMPTER2, COMPTER2 :nombre+1
[2] In COMPTER2, if :nombre > 5 [stop]
[2] In COMPTER2, COMPTER2 :nombre+1
[3] In COMPTER2, if :nombre > 5 [stop]
[3] In COMPTER2, COMPTER2 :nombre+1
[4] In COMPTER2, if :nombre > 5 [stop]
[3] In COMPTER2, pr :nombre - Réponse: 5
[2] In COMPTER2, pr :nombre - Réponse: 4
[1] In COMPTER2, pr :nombre - Réponse: 3
```

Dans ces deux versions, on atteint d'abord à chaque étape le niveau suivant dans l'appel de procédure. Dans COMPTER1, la ligne d'impression est exécutée avant l'auto-appel. Les nombres sortis apparaissent de ce fait en ordre croissant. Une fois que la condition d'interruption est remplie, l'exécution de COMPTER1 se termine.

Pour le traitement de COMPTER2, l'exécution du niveau actuellement atteint est interrompue lors de la récursion et on passe alors au prochain niveau dans l'appel de procédure. Une fois la condition d'interruption remplie, on revient étape par étape au niveau de départ de COMPTER2.

La ligne d'impression est alors sortie (et indiquée par trace) par trois fois successives en commençant par la ligne d'impression rencontrée avant que la condition d'interruption ait été remplie.

Comme l'indique le chiffre entre crochets au début de la ligne de contrôle, on passe à un niveau supérieur lors de chaque appel et on revient ensuite progressivement au niveau de départ. Cela ne se produisait pas dans COMPTER1.

Pour pouvoir sortir ici la valeur de nombre, il faut qu'on puisse disposer de la valeur qui s'applique pour chaque niveau. Cela n'est possible que si le LOGO a conservé la valeur correspondante lors de l'interruption de la procédure COMPTER2 par l'auto-appel. Dans l'exemple très simple que nous avons pris, il n'y a qu'un nom à stocker mais, plus généralement, ce sont tous les noms locaux qui

doivent être stockés dans une pile prévue à cet effet lorsque l'exécution d'une procédure est interrompue. Une fois que la place réservée à cet effet est épuisée, la récursion ne peut se poursuivre et le système LOGO réagit par le message Out of LOGO stack space!

L'inversion de l'ordre dans lequel sont sortis les nombres dans COMPTER1 et COMPTER2 vient du fait que la ligne d'impression est sortie dans COMPTER1 lors de la progression dans la récursion alors que dans COMPTER2 les nombres sont sortis par contre lors de la régression vers le niveau précédent de l'appel de procédure.

13.3 Fonctions récursives

Classements de différents objets

Combien y a-t-il de possibilités pour ranger onze livres différents sur une étagère? La réponse est 39916800. La réponse suivante est plus concrète: il y a onze fois plus de possibilités de classer onze livres que d'en classer dix. Pour tout classement donné de dix livres, le onzième pourra être placé devant le premier, le seconde, etc... ou encore après le dizième livre. Il y a donc onze possibilités d'insérer un onzième livre dans tout classement de dix livres.

Cette réponse ne nous fournit pas de nombre mais elle nous permet de mieux comprendre comment trouver une solution générale à ce problème pour un nombre quelconque d'objets. Elle conduit directement à une solution récursive:

> Pour déterminer le nombre de classements possibles de N objets, il faut calculer le nombre de classements pour (N-1) objets.

```
to CLASSEMENTS :n
op :n * CLASSEMENTS :n-1
end
```

Pour le problème posé, il faudra calculer une valeur en sortie. C'est pourquoi la procédure correspondante doit sortir une valeur avec op. L'appel de procédure devra alors figurer dans une ligne de programme là où un mot peut figurer; par exemple à la place d'un nombre ou de la valeur d'un nom. Les procédures récursives avec valeur en sortie doivent aussi être utilisées de cette manière. La procédure CLASSEMENTS est appelée de façon récursive dans la ligne dans laquelle se produit la sortie avec op. L'auto-appel fait donc partie de l'entrée de op.

La procédure CLASSEMENTS n'est pas encore achevée car la récursion ne s'arrête pas pour le moment. Et de fait, avec cette version, le LOGO vous enverra vite le message d'erreur Out of LOGO stack space! Bien que l'auto-appel figure à la fin, il ne s'agit pas de la forme simple de la récursion qui autorise autant de répétitions qu'on le souhaite. La raison en est facile à comprendre: le résultat, ici le nombre de classements, doit être conservé pour la multiplication car celle-ci ne peut être exécutée qu'une fois que la récursion est terminée.

Pour le calcul récursif d'une valeur, il faut une condition d'interruption. Comme un résultat est nécessaire, il faut qu'une valeur en sortie existe réellement lors de l'arrêt de la récursion. Un calcul récursif ne peut fonctionner que lorsqu'on connaît une valeur spéciale avec laquelle on puisse mettre fin à la récursion. Dans le cas présent, la récursion pourra être interrompue lorsqu'on atteindra un objet unique auquel cas il n'y a bien sûr qu'une possibilité de classement.

```
to CLASSEMENTS :n
if :n=1 [op 1]
op :n * CLASSEMENTS :n - 1
end
```

La fonction classements que nous avons décrite ici correspond à ce qu'on appelle le calcul des probabilités en mathématiques.

Combien y a-t-il de combinaisons possibles au loto?

Avec le loto de six nombres sur 49, il y a 13983816 combinaisons possibles. Ce nombre peut être calculé de façon récursive avec le raisonnement suivant:

On ne tire d'abord que cinq nombres. Il reste alors 49-5 = 44 possibilités pour le sixième nombre. Il devrait donc y avoir (49-5) fois plus de possibilités pour le choix de six nombres que pour le choix de cinq nombres. En réalité, il y a moins de possibilités parce qu'une combinaison donnée de six nombres peut provenir de différentes combinaisons de cinq nombres. La combinaison (1,5,13,24,35,46) peut être produite à partir de toutes les combinaisons de cinq nombres faisant partie de cette combinaison. Il y a donc exactement six combinaisons de cinq nombres valables. On en déduit donc que:

Il y a (49-5)/6 fois plus de combinaisons de six nombres que de combinaisons de cinq nombres, (49-4)/5 fois plus de combinaisons de cinq nombres que de combinaisons de quatre nombres. Cela peut être facilement exprimé par un mot LOGO approprié:

```
to COMBINAISONS :n
if :n=1 [op 49];49 possibilités pour un nombre
op (50-:n)/:n * COMBINAISONS :n - 1
end
```

Essayez:

```
COMBINAISONS 6
```

Réponse: 13983816

Cette procédure peut être aisément généralisée pour n'importe quel loto avec tirage de n nombres parmi m:

```
to COMBINAISONS :n :m
if :n=1 [op :m];m possibilités pour un nombre
op (:m+1 - :n)/:n * COMBINAISONS :n - 1
end
```

13.4 Autres exemples de récursion

Calcul de polynôme

Le calcul des valeurs d'un polynôme d'après le schéma de Horner, tel que nous l'avons étudié à la leçon 10, constitue un exemple de fonction définie de manière récursive.

```
to HORNER :x :a
if emptyp :a [op 0]
op (first :a) + :x * HORNER :x bf :a
end
```

La récursion concerne ici la liste a des coefficients. Le résultat est calculé à partir de la valeur du polynôme obtenu lorsqu'on laisse de côté le premier coefficient.

PGDC et PPMC: algorithme d'Euclide

PGDC: plus grand diviseur commun
PPMC: plus petit multiple commun

Ces termes vous rappellent peut-être de mauvais souvenirs de calcul de fractions. Avec le LOGO, tout cela devient très simple! La méthode portant le nom d'Euclide permet de déterminer de manière élégante le plus grand diviseur commun de deux nombres. On prend le nombre le plus grand des deux et on détermine le reste de la division du plus grand par le plus petit:

```
54=12*4 + 6
```

Le reste de la division et le nombre le plus petit ont le même PGDC que les valeurs de départ mais la paire (12,6) contient des nombres plus petits que (54,12). L'opération se termine lorsqu'il ne reste plus de reste. Cet algorithme peut être formulé très facilement de façon récursive.

```
to PGDC :a :b
if :b = 0 [op :a]
```

```
op PGDC :b remainder :a :b
end
```

Cette procédure fonctionnera également si le premier nombre est plus petit que le second. Dans ce cas, les deux nombres seront échangés lors de la première étape de l'opération. Le plus petit multiple commun sera alors obtenu avec:

```
to PPMC :a :b
op :a * :b / PGDC :a :b
end
```

Le tétraèdre, une récursion graphique

```
to TRIANGLE :cote
rt 150 fd :cote * 1.732
rt 150 fd :cote
rt 180
end
```

Essayez:

```
fd 80 TRIANGLE 80
```

Vous obtenez un triangle de côtés égaux. Répétez maintenant cette ligne une seconde fois! Vous obtenez à nouveau deux triangles égaux qui ensemble donnent l'impression d'un tétraèdre.

La procédure TRIANGLE ne dessine que deux côtés de triangle. Cela correspond bien sûr à une intention particulière. Une fois que le premier côté du triangle a été dessiné, une interruption peut avoir lieu et une autre figure pourra être représentée dans l'emplacement actuel. Nous allons utiliser ici de façon récursive la procédure de projection d'un tétraèdre.

```
to TETRA :cote :e
if :e = 0 [stop]
repeat 3 [fd :cote TETRA :cote/1.9 :e-1 TRIANGLE :cote]
```

```
end
```

La grandeur e indique le niveau de l'imbrication. Essayez:

```
cs ht bk 30 TETRA 90 4
```

Un programme simple qui produit le dessin de la figure 13.1!

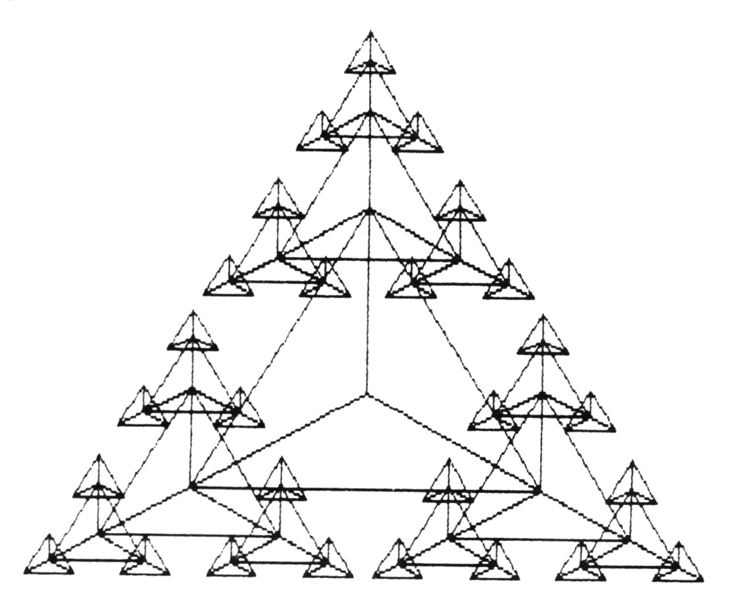


Figure 13.1: Le tétraèdre de quatre niveaux

POCALL vous permettra de constater la simplicité de la structure employée:

```
POCALL "TETRA
0 : TETRA
1 : ... TETRA*
2 : ... TRIANGLE
```

Nous vous proposons d'observer un autre exemple d'arbre produit par récursion. Nous ne décrirons pas le programme plus avant.

```
to ARBRE :longueur :angle :n
if :n = 0 [stop]
fd :longueur
lt :angle ARBRE :longueur :angle :n - 1
rt 2*:angle ARBRE :longueur * .9 :angle :n - 1
lt :angle bk :longueur
end
```

Appel: `cs pu setpos [30 - 170] pd ARBRE 30 17 9`

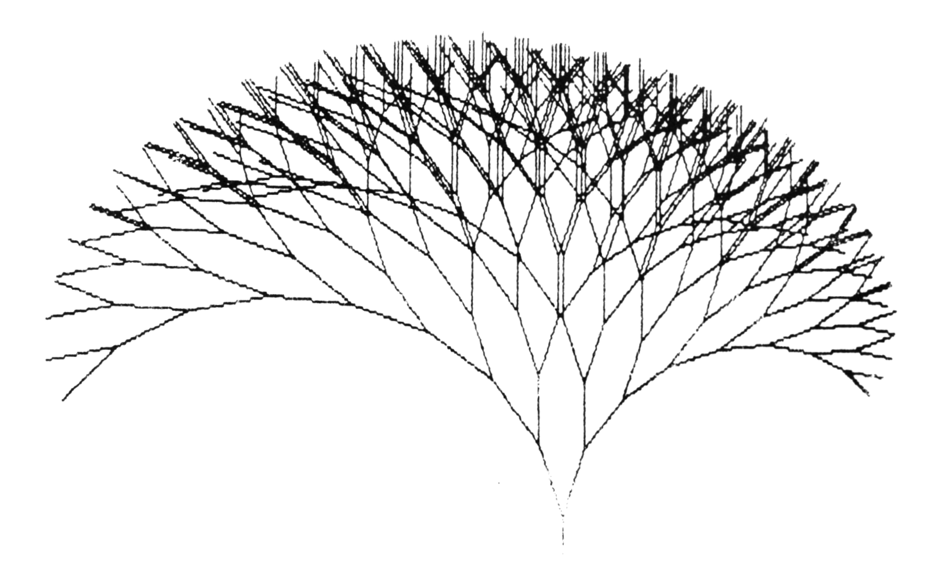


Figure 13.2: Un arbre binaire produit par récursion

13.5 Les tours de Hanoï

Le jeu mathématique inventé par le mathématicien Lucas est un exemple classique de récursion. On empile des disques de différents diamètres, d'après leur taille, les plus petits en haut. A côté, il y a encore deux emplacements où les disques peuvent être empilés. Le but est de déplacer la tour pour l'amener dans l'un des deux autres emplacements. Mais pour ce faire, les disques doivent

toujours être placés soit dans un emplacement vide, soit sur des disques de plus grand diamètre.

L'ordinateur peut ici remplacer le matériel de jeu en indiquant à tout moment à l'écran la situation des trois emplacements. Les procédures auxiliaires que nous vous présentons ci-dessous permettent de formuler un tel programme de jeu.

Le but est ici de programmer la stratégie de jeu idéale.

Avec deux disques seulement, il est facile d'indiquer les coups nécessaires pour déplacer la tour de l'emplacement 1 vers l'emplacement 2:

```
1 vers 3 / 1 vers 2 / 3 vers 2
```

Avec trois disques, on déplacera d'abord la partie de la tour au-dessus du disque inférieur vers l'emplacement 3. Nous savons déjà comment déplacer une tour de deux disques. Le disque inférieur de la première tour sera alors placé dans l'emplacement 2 vers lequel nous voulons déplacer la tour. La pile de deux disques devra ensuite être déplacée du troisième vers le second emplacement ce qui nous ramène au problème déjà résolu du déplacement d'une tour de deux disques.

```
1 vers 2 / 1 vers 3 / 2 vers 3
1 vers 2
3 vers 1 / 3 vers 2 / 1 vers 2
```

Avec n disques, la partie supérieure de (n-1) disques sera amenée dans le troisième emplacement qui n'est pas l'emplacement définitif. Le disque inférieur sera alors amené dans l'emplacement définitif et la tour de (n-1) disques sera déplacée de son emplacement de stockage vers son emplacement définitif. Ainsi parvient-on à une solution récursive qui utilisera chaque fois l'action déplaçant une tour d'un nombre de disques inférieur de 1. La récursion se terminera lorsqu'il n'y aura plus qu'un disque à déplacer.

L'action de base consiste à déplacer un disque d'un emplacement

dans un autre. On peut sortir les indications correspondantes sans sortie graphique et concrétiser le jeu avec de vrais disques qu'on déplacera d'après les instructions du programme LOGO. Les procédures de représentation graphique seront indiquées un peu plus loin.

```
to TRANSFERER :de :vers
(pr [DEPLACEZ] :de [ VERS EMPLACEMENT ] :vers)
end
```

La solution véritable est programmée dans la procédure DEPLACER.

```
to DEPLACER :n :de :vers
if :n = 1 [TRANSFERER :de :vers stop]
local "placerangement
make "placerangement word "l 6 - (last :de) - (last :vers)
DEPLACER (:n - 1) :de :placerangement
TRANSFERER (:n - 1) :de :vers
DEPLACER (:n - 1) :placerangement :vers
end
```

Les tours ne sont pas indiquées simplement par leur numéro mais par l1, l2, l3. Cette méthode est très utile pour la gestion des disques. Une gestion programmée n'est pas encore nécessaire ici mais les programmes pourront continuer à être utilisés lorsqu'il s'agira de les compléter par la représentation graphique. last isole le numéro du nom alors qu'inversement word produit à nouveau le nom qui convient à partir du numéro.

Le numéro de l'emplacement de rangement est déterminé dans la troisième ligne du programme, à l'aide de la formule indiquée à cet endroit. On peut facilement s'assurer que cela permet bien de déterminer l'emplacement réellement libre et non de ou vers. Les autres lignes restituent la procédure décrite plus haut. Il nous manque encore un programme-cadre.

```
to TOUR :nombre
type [DE L'EMPLACEMENT 1 A L'EMPLACEMENT?]
DEPLACER :nombre "l1 word "l rq
```

L'examen avec POCALL nous donnera :

```
POCALL "TOUR
0 : TOUR :NOMBRE
1 : ... DEPLACER
2 : ...... TRANSFERER*
2 : ...... DEPLACER
2 : ...... TRANSFERER*
```

Procédures graphiques pour les tours de Hanoï

Les procédures suivantes peuvent être utilisées pour représenter graphiquement sur l'écran la solution optimale du jeu. Les programmes graphiques sont plus compliqués que la programmation du jeu lui-même.

Avec les procédures PLACER, SUR, RETIRER, vous pouvez pour ainsi dire dessiner un rectangle plein représentant un disque dans un endroit déterminé ou bien placer ce disque sur une pile ou encore l'en retirer.

```
to PLACER :n :g :p
if :n < 1 [stop]
pu setpos se (200 * last :p) - 400 - :g/2 22*:n - 100
DISQUE :g
end

to DISQUE :gr
type char 7;Ton
px seth 90
repeat 2 [fd :gr lt 90 fd 16 lt 90]
end

to SUR :p :g
make :p se thing :p :g
pd
PLACER count thing :p :g :p
end

to RETIRER :p
```

```
(local "l "g)
make "l thing :p make "g last :l
pe
PLACER count :l :g :p
make :p bl :l
op :g
end
```

Dans PLACER, n indique la position du disque sur la pile, g la taille du disque et p le numéro d'emplacement. On place ainsi un disque dans un emplacement bien déterminé qui est calculé dans la troisième ligne.

Le disque lui-même est dessiné à l'aide de la procédure DISQUE. Le caractère ASCII 7 (bell) est d'abord sorti pour produire une signal acoustique. Le crayon à dessin est ensuite placé avec

```
px
```

dans un état qui se traduit par le fait que la tortue inverse les couleurs lors du dessin, de sorte qu'un rectangle sera dessiné lorsque l'écran graphique y est effacé. Un appel ultérieur efface à nouveau le rectangle. Essayez:

```
DISQUE 100 repeat 200 [] DISQUE 100
```

SUR sert à placer sur la pile de numéro p un autre disque de taille g. RETIRER supprime le disque le plus élevé d'une pile. Ces deux actions utilisent bien sûr pour cela l'action PLACER. Il faut encore tenir un registre de l'état des trois emplacements. En effet, pour la représentation graphique des tours, il faut savoir combien de disques et quels disques se trouvent chaque fois dans un emplacement donné. Ce registre est tenu à l'aide des listes l1, l2 et l3.

Sous LOGO, on peut fabriquer soi-même des noms à l'intérieur du programme comme nous le faisons ici pour l1, l2 et l3 à partir de la lettre l et des chiffres 1, 2 et 3. Lorsque le numéro de pile est disponible sous le nom p, le nom correspondant est produit avec word "l :p. Si on veut ensuite obtenir la valeur de ce nom, il

faudrait placer un double point devant ce nom, d'après la syntaxe habituelle du LOGO. Mais cela ne suffit pas car la valeur sera alors le nom complet tel qu'il vient d'être formé. Cela tient au fait que le nom n'est pas fourni directement mais qu'il est lui-même dérivé d'une autre valeur. La valeur, en l'occurrence la valeur de la liste l1 (ou l2 ou l3) peut être déterminée au moyen du mot LOGO thing.

Lorsqu'un disque est placé sur une pile, il faut compléter la liste correspondante en y ajoutant la taille du disque nouvellement posé. Lorsqu'un disque est retiré d'une pile, il faut au contraire supprimer cette indication de la liste correspondante. Cette indication sera renvoyée comme résultat car le disque devra alors être placé sur une autre pile.

Le transfert est maintenant réalisé graphiquement simplement par RETIRER et SUR:

```
to TRANSFERER :de :vers
SUR :vers RETIRER :de
end
```

Le résultat de RETIRER sert de valeur d'entrée pour SUR. Pour parachever notre oeuvre, il ne nous manque plus que les procédures TOUR et CONSTRUIRE qui se chargeront de présenter et de dessiner la situation de départ du jeu.

```
to TOUR :nombre
ht cs
make "l1 [] make "l2 [] make "l3 []
CONSTRUIRE :nombre 20 "l1
type [de l'emplacement 1 a l'emplacement ?]
DEPLACER :nombre "l1 word l rq
end

to CONSTRUIRE :n :g :p
if :n =0 [stop]
PLACER :n :g :p
make :p se :g thing :p
CONSTRUIRE :n - 1 :g + 20 :p
```

```
end
```

Essayez d'abord ce programme avec de petites tours et observez ensuite l'accroissement du temps d'exécution au fur et à mesure de l'augmentation du nombre de disques.



Figure 13.3: Une situation en milieu de jeu.

Vous pouvez examiner le fonctionnement de tout le paquet de procédures pour les tours de Hanoï à l'aide de POCALL (programmes donnés à la leçon précédente). L'analyse livrera le résultat suivant:

```
0 : TOUR
1 : ...CONSTRUIRE
2 : ......PLACER
3 : .........DISQUE
2 : ......CONSTRUIRE*
1 : ...DEPLACER
2 : ......TRANSFERER
3 : .........SUR
4 : ............PLACER
```

```
5 : ...............DISQUE
3 : .........RETIRER
4 : ............PLACER
5 : ...............DISQUE
2 : ......DEPLACER*
2 : ......TRANSFERER
2 : ......DEPLACER
3 : .........SUR
4 : ............PLACER
5 : ...............DISQUE
3 : .........RETIRER
4 : ............PLACER
5 : ...............DISQUE
2 : ......DEPLACER*
```

Confrontation des solutions itératives aux solutions récursives

En dehors de la récursion employée comme pure méthode de répétition, il faut bien dire que la compréhension des solutions récursives pose des problèmes à bon nombre d'utilisateurs. C'est pourquoi la récursion a été expliquée ici de façon aussi détaillée. Nous vous invitons à prendre à l'occasion le temps nécessaire pour bien assimiler le déroulement des programmes récursifs que nous vous avons présentés comme exemples.

Lorsque le problème à résoudre est complexe de par sa nature même, il arrive souvent que les projets de solutions récursives soient beaucoup plus faciles à imaginer que des solutions itératives. Il est donc tout à fait possible de procéder de la façon suivante: rechercher d'abord une solution récursive de façon à disposer au moins d'une solution quelconque. Puis essayer ensuite de découvrir des méthodes itératives car celles-ci présentent en règle générale l'avantage de travailler plus vite.

A titre d'exemple, nous allons nous servir une nouvelle fois des tours de Hanoï.

Vous aurez certainement du mal au départ à découvrir directement une méthode itérative de solution. Il vous faudra en effet développer pour cela un algorithme spécial. Observez alors ce qui

se passe avec la méthode récursive. Vous constaterez bientôt qu'une étape sur deux consiste à déplacer le plus petit de tous les disques. Vous n'avez plus ensuite de choix pour le coup suivant si vous voulez éviter de répéter les mêmes coups. On ne peut rien poser sur l'emplacement où se trouve le plus petit disque. Il ne reste plus alors qu'une seule possibilité pour les deux emplacements de tours restants car un des deux disques placés au sommet de chacune des deux tours est plus petit que l'autre.

Il ne vous reste plus maintenant qu'à surveiller le déplacement du plus petit disque! Il s'avère que celui-ci avance toujours dans la même direction pour une situation de départ et un but de transfert donnés. Il faut donc se représenter les trois tours comme si elles étaient disposées en cercle. Le plus petit disque devra donc évoluer soit dans le sens des aiguilles d'une montre, soit dans le sens contraire, en fonction du nombre de disques et du but fixé. L'opération doit donc se dérouler de la manière suivante:

- Déplacement du plus petit disque d'une unité, par exemple vers la droite.

- Rechercher lequel des deux autres disques placés au sommet d'une tour doit être déplacé.

- Tester si l'opération est terminée.

Une fois considérées les étapes essentielles de l'opération, la solution apparaît très simple. Mais toute la difficulté consiste justement à bien analyser les différentes étapes nécessaires. C'est ce qu'on appelle la recherche d'un 'algorithme'.

```
to STRATEGIE :t1 :t2 :t3
;SOLUTION ITERATIVE POUR LES TOURS DE HANOI
label "debut
make "t3 :t1 make "t1 :t2
TRANSFERER :t3 :t2; DEPLACEMENT DU PLUS PETIT DISQUE
make "t2 SUIVANT :t1
if emptyp thing :t3 [TRANSFERER :t2 :t3 go "debut]
if emptyp thing :t2 [TRANSFERER :t3 :t2 go "debut]
if LARGEUR :t3 > LARGEUR :t2 [TRANSFERER :t2 :t3 go "debut]
```

```
TRANSFERER :t3 :t2 go "debut
end

to SUIVANT :l
op item last :l [l2 l3 l1]
end

to LARGEUR :l
op last thing :l
end
```

La procédure STRATEGIE remplace, dans la solution itérative, la procédure DEPLACER. La dernière ligne de TOUR devra donc être ainsi modifiée:

```
STRATEGIE "l1 "l2 "l3
```

STRATEGIE utilise les deux programmes auxiliaires SUIVANT et LARGEUR. SUIVANT sort le nom de la tour suivante en suivant un tour de rôle cyclique. Pour l1, l2 est sorti, pour l2, l3 et pour l3, l1. LARGEUR donne la largeur du disque placé au sommet de la pile considérée.

On déplace d'abord le disque le plus petit et on détermine le but avec la procédure SUIVANT. Au coup suivant, on examine les deux tours sur lesquelles ne se trouve pas le disque le plus petit. Si l'une de ces deux tours est vide, le coup à jouer est évident. Dans le cas contraire, il faut examiner sur le sommet de laquelle des deux tours figure le disque le plus large. Cela se fait grâce à LARGEUR.

La solution itérative proposée ici n'est pas encore aussi générale que la solution récursive car on ne peut pas choisir si la tour doit être transférée dans le second ou le troisième emplacement. Le but du transfert est en effet décidé en fonction du nombre de disques. Si vous voulez définir auparavant le but du transfert, le programme doit être rendu encore plus général en déterminant auparavant dans quel sens le plus petit disque doit tourner. Cette généralisation du programme pourrait constituer pour vous un excellent exercice.

Il subsiste encore une autre faute de goût: une fois le transfert terminé, le programme se termine par un message d'erreur car nous n'avons encore prévu aucune interruption de la répétition avec l'instruction de saut go. Le plus simple est d'apporter une petite modification à la procédure RETIRER:

```
to RETIRER :p
local "l make "l thing :p
if emptyp :l [throw "TOPLEVEL]
...
```

L'instruction throw "TOPLEVEL constitue une nouveauté. Elle provoque la fin du programme, comme stop, mais sans saut en retour au programme d'appel. L'ensemble du déroulement du programme est interrompu et le système est ensuite à nouveau prêt à recevoir vos entrées.

Exercices

1) Observez combien de fois le programme doit déplacer des disques pour transférer une tour de Hanoï de 2, 3, 4, etc... disques!

2) Un capital initial c devra produire un pourcentage p d'intérêts. Ecrivez une procédure récursive qui sortira en résultat la valeur du capital augmenté des intérêts au bout de n années (ou périodes d'échéance des intérêts).

3) Un épargnant verse annuellement une somme r sur un compte rémunéré à un taux d'intérêts p. Formulez à l'aide de la récursion un programme qui sortira l'état du compte au bout de n années.

4) Généralisez la solution itérative des tours de Hanoï de façon à ce que l'utilisateur du programme puisse choisir le but du transfert de la tour. Il vous faudra formuler, en plus de la procédure auxiliaire SUIVANT, une procédure PRECEDENT analogue.

Leçon 14: Programmation graphique avancée

Nouveaux éléments de vocabulaire:
dot, dotc, fill

Programmes:
STAMPWORD, TRIANGLE (cas ccc), ONDE, C, DRAGONG, DRAGOND, HILBERT, SIERPINSKI, BASCULE, ANNEAUX, FAUCILLE, FUITE, DE, POINT, VECTEUR, TETRAEDRE

14.1 Mélange de textes et de dessins

Lorsqu'un caractère doit être sorti sur l'écran de texte, une image toute prête pour la représentation du caractère doit figurer quelque part dans une grille de 8 points sur 8. Les images utilisées sont empruntées au jeu de caractères actuellement disponible qui permet une sélection parmi 224 images possibles. Vous pouvez examiner le jeu de caractères avec la ligne

```
make "i 32 repeat 224 [type char :i make "i :i+1]
```

Les opérations de dessin consistent au contraire à accéder à des points isolés de l'écran graphique. Si le choix de caractères offert vous semble déjà largement suffisant, sachez que ce n'est rien par rapport au nombre total des variantes possibles sur une grille de 8 points sur 8 (même pour une représentation en une couleur, ce nombre atteint 20 chiffres décimaux!).

Il y a donc des différences fondamentales entre la production de dessins et la sortie de caractères de texte. C'est pourquoi certains microordinateurs gèrent, de façon interne, les écrans graphique et de texte comme des zones totalement séparées. Sur les ordinateurs AMSTRAD, les deux écrans font partie de l'écran global si vous avez besoin simultanément de texte et de dessin. C'est pourquoi on parle aussi d'une division entre fenêtres graphique et de texte.

Le mélange de textes et de dessins n'est malheureusement pas soutenu, sous DR LOGO, par des instructions de base et il faut donc avoir en partie recours à des astuces.

Le texte dans la fenêtre graphique sur le PCW AMSTRAD

La sortie de texte sur l'écran graphique est très simple sur le PCW puisque cela ne pose aucun problème particulier d'amener le curseur de texte dans la fenêtre graphique. Si vous entrez une instruction graphique telle que home par exemple, vous vous retrouvez de façon standard dans un mode avec écran divisé. La position du curseur de texte peut être obtenue avec l'instruction

```
cursor
```

Résultat (par exemple): [0 22]

Ce résultat indique le début de la 22ème ligne; c'est la ligne dans laquelle la réponse a également été sortie, c'est-à-dire la seconde ligne de la fenêtre de texte. Mais on peut cependant également amener le curseur de texte dans la fenêtre graphique.

```
setcursor [20 10] type [Voici le curseur de texte]
```

Vous voyez donc tout de suite comment mélanger des textes et du graphisme sur le PCW:

Le curseur de texte doit être amené dans l'emplacement souhaité de la fenêtre graphique. La sortie est alors effectuée comme dans la fenêtre de texte.

Une fois que LOGO revient en mode direct, le curseur et le point d'interrogation (interrogation du système) apparaissent sur la ligne suivant la position du curseur de texte atteinte en dernier lieu, souvent donc en plein dans la fenêtre graphique. Pour ramener le curseur dans la fenêtre de texte, le plus simple est de vider la fenêtre de texte avec ct.

```
to TEXEMPLE
cs setcursor [20 10] pr [Exemple test]
repeat 100 [fd random 80 rt random 360]
ct;Ramener le curseur de texte
end
```

Si le vidage de l'écran n'est pas souhaité, on peut noter au début la situation actuelle du curseur de texte pour la rétablir à la fin:

```
make "textcursor cursor ... setcursor :textcursor
```

On parvient par exemple, sur le PCW, sur l'origine du système de coordonnées, c'est-à-dire dans la position atteinte avec home, avec la position de curseur de texte [45 15]. Verticalement il faut avancer de 16 pas et horizontalement de 16 pour avancer d'un caractère.

Mélange de textes et de dessins sur le CPC AMSTRAD

Sur le CPC, le curseur de texte ne peut pas être amené dans la fenêtre graphique, une tentative dans ce sens risquant de produire des effets indésirables dans la fenêtre de texte.

Le CPC dispose cependant d'un curseur graphique qui est naturellement déterminé sous LOGO par la position de la tortue et qui peut donc être modifié par des instructions pour la tortue. Un caractère peut être sorti dans cet emplacement.

```
to STAMPWORD :mot :m if emptyp :mot [stop]
type (word char 23 :m char 5 first :mot)
STAMPWORD bf :mot :m
end
```

La sortie dans l'emplacement du curseur graphique est obtenue à travers le caractère de commande placé devant le mot par la troisième ligne char 5 de STAMPWORD. La procédure STAMPWORD permet de sortir un mot entier. En effet, lors de sorties de caractères se suivant immédiatement les unes les autres, les

caractères sont placés les uns à la suite des autres sans qu'il soit nécessaire de déplacer la tortue.

Avant char 5 dans la troisième ligne, le caractère de commande 23 est encore inséré dans la sortie ainsi que la valeur entrée pour m. m représente ici le mode de dessin, zéro étant la valeur normale.

```
cs STAMPWORD "Salut 0
```

L'entrée du mode de dessin est nécessaire car le système LOGO sélectionne automatiquement le mode 1, ce qui se traduit par des couleurs de dessin inversées ou différentes lors d'un changement de couleurs.

```
cs STAMPWORD "Salut 0 fd 0 STAMPWORD "Salut 1
```

Cette ligne vous montre que fd 0 a pour effet de ramener le curseur graphique dans l'emplacement déterminé par la position de la tortue. Lors des déplacements de la tortue, la tortue doit être visible pour STAMPWORD pour que la modification du curseur graphique soit évaluée correctement.

La sortie suivante de Salut entraînera maintenant l'effaçage du mot si la couleur de dessin reste la même. Lorsque plusieurs séquences de caractères sont superposées, c'est souvent de cette façon qu'on procédera pour rendre l'écriture précédente illisible.

Il est naturellement assez surprenant pour un programmeur LOGO d'avoir recours directement aux codes de commande de la machine. Le fabricant aurait certainement mieux fait de prévoir ici un mot LOGO particulier. Dans le manuel d'utilisation, le mélange de textes et de dessins n'est même pas évoqué.

14.2 Fixation et suppression de points

En graphisme avec la tortue, un dessin est normalement créé par la trace que la tortue laisse sur l'écran lors de ses déplacements.

La version du LOGO qui nous intéresse ici offre cependant également

des instructions permettant de produire sur l'écran des points isolés. On utilise pour cela les mots LOGO dot et dotc.

```
dot [200 100]
```

Un point sera ainsi fixé, dans la couleur actuelle du crayon à dessin, dans l'emplacement défini par l'entrée. Essayez:

```
repeat 1000 [dot se -320+random 640 -120+random 320]
```

La ligne

```
dot tf
```

fixera un point dans l'emplacement actuel de la tortue. On peut aussi utiliser pour cela l'instruction

```
fd 0
```

ce qui dans ce cas précis est naturellement plus simple. L'instruction dot peut d'ailleurs être en principe remplacée systématiquement par

```
setpos [200 100] fd 0
```

Le mot LOGO dotc fournit comme résultat le numéro de la couleur du point entré.

```
setpc 1 dot [100 100] dotc [100 100]
```

Résultat: 1

Le fond est réglé par défaut sur la couleur 0 si vous ne le modifiez pas avec setbg. Vous pouvez ainsi utiliser dotc pour tester des points isolés.

Sur le PCW, dotc indique directement si un point est mis ou non sur la fenêtre graphique puisque le fond a ici toujours le code de couleur 0 et que les points mis ont toujours le numéro de couleur 1. Sur le CPC, on peut déterminer la couleur du fond à l'aide de sf

si celle-ci n'est pas connue dès le départ.

```
if dotc [100 100] = 1 [(pr "Point [100 100] [marque)]
if not dotc [100 100] = first sf [ ... ]
```

dotc permet également de déterminer si le point appelé figure sur l'écran car dotc renverra la valeur -1 comme résultat si les coordonnées sortent du cadre de l'écran graphique.

Points d'intersection: Constructions triangulaires

L'emploi de dotc comme instruction de test de points isolés permet également de déterminer si le crayon à dessin rencontre des lignes déjà existantes. On peut ainsi déterminer sur le plan graphique des points d'intersection de droites et de courbes. Cela peut déboucher sur de nombreuses applications en mathématiques.

Nous allons prendre comme exemple la construction d'un triangle à partir des trois longueurs de côtés. Ce problème sera résolu sur le papier de la façon suivante: un côté sera dessiné puis on recherchera ensuite un point d'intersection de deux cercles. La fonction LOGO dotc nous permettra d'entreprendre cette construction sur l'écran.

```
to TRIANGLE :a :b: :c
ht (local "ak "bk "ck)
make "ak se -:c/2 -80;Point A
make "bk se :c/2 -80;Point b
pu setpos :bk pd setpos :ak;Côté c
DEMICERCLE :b;cercle autour du point A de rayon b
pu setpos :bk
make "ck COUPECERCLE :a;cercle autour du point B de rayon
a
if emptyp :ck [pr [pas de solution] stop]
pd setpos :bk setpos :ak setpos :ck
(pr "Gamma: (towards :ak) - (towards :bk))
setpos :ak (pr "Alpha: (towards :bk) - (towards :ck))
setpos :bk (pr "Beta: (towards :ck) - (towards :ak))
end
```

```
to COUPECERCLE :r
(local "u "w)
pu seth -90 fd :r rt 90
make "u 1.5708*:r make "w 180/:u rt :w/2
repeat 4 [fd 2 rt :w]
repeat :u [fd 2 rt :w if or (dotc tf >0) (dotc se first !
tf 1+item 2 tf>0) [type char 7 op tf] [op []]
end

to DEMICERCLE :r
pu seth 90 fd :r lt 95 pd
local "s make "s .17453 * :r
repeat 18 [fd :s lt 10]
pu lt 85 fd :r
end
```

La procédure auxiliaire DEMICERCLE dessine un demi-cercle autour du point A. La tortue est ensuite conduite sur un second demi-cercle autour du point B à l'aide de COUPECERCLE. Le crayon à dessin reste relevé pour cela car l'instruction de test dotc ne doit identifier que les points du cercle dessiné en premier.

Alors que pour le premier demi-cercle on ne dessine en réalité que la moitié du polygone régulier de 36 côtés, la tortue doit avancer très progressivement la seconde fois pour que le point d'intersection puisse vraiment être atteint. Nous avons choisi ici un pas de 2 (fd 2). dotc est ensuite utilisé avec les valeurs de coordonnées, qui viennent d'être atteintes et qui peuvent être déterminées avec tf, et avec le point de coordonnée y immédiatement supérieure pour tester si on est parvenu au point d'intersection. (Sur le PCW il vaut mieux augmenter de deux.)

La procédure COUPECERCLE fournit en cas de succès, accompagnées d'un bip, les coordonnées du point d'intersection. En cas d'échec, par contre, une liste vide est renvoyée comme résultat.

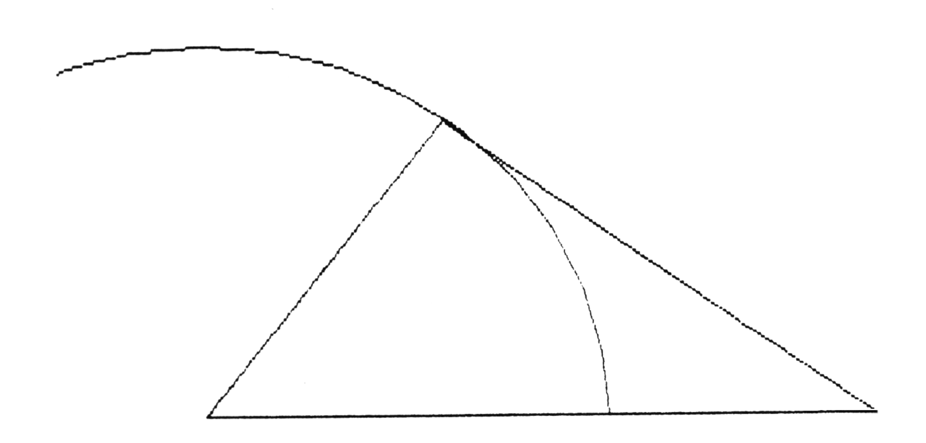


Figure 14.1: Construction d'un triangle

La solution du problème de construction est finalement utilisée également pour déterminer les angles du triangle. On vise pour cela à partir d'un coin les deux autres coins à l'aide du mot LOGO towards et on détermine ainsi l'angle recherché.

14.3 Répartition de différentes densités

On peut produire des dessins intéressants avec des droites, des polygones ainsi qu'avec des parcelles remplies, coloriées ou hachurées. Il est cependant également possible d'utiliser la répartition de points isolés à des fins graphiques. Les points seront généralement dispersés de manière irrégulière à l'aide de nombres aléatoires. Cela peut produire des effets intéressants en donnant par exemple une impression de relief sous différents éclairages.

Nous nous contenterons ici d'expliquer le principe par des exemples

simples. Une répartition aléatoire régulière de points est bien sûr sans grand intérêt. Les choses sérieuses ne commencent qu'à partir du moment où on fixe beaucoup de points dans certaines zones et moins dans d'autres.

La fenêtre graphique est une construction en deux dimensions. Pour parvenir à différentes densités de points, il faut qu'un nombre correspondant à chaque point image possible décide si un point doit être fixé. Pour obtenir une dispersion des points au hasard, le nombre affecté à chaque point sera comparé à un nombre tiré au sort. Si on décide alors de fixer un point chaque fois que le nombre tiré au sort sera plus petit que la valeur affectée au point considéré, il y aura une plus grande probabilité de présence de points isolés dans les zones auxquelles sont affectés des nombres élevés.

Des répartitions de points isolés de ce type sont donc fixées par l'indication d'une fonction qui affecte une valeur numérique à tout point image possible de l'écran graphique. Comme les nombres aléatoires produits avec random ne sont pas négatifs, il faudra que les valeurs de la fonction soient positives. La valeur la plus élevée possible devra correspondre à la valeur utilisée comme entrée lors de l'appel de random. Cela peut toujours être obtenu à l'aide de facteurs appropriés. Les zones du dégradé éventuel de l'image produite seront déterminées par la zone de dispersion des nombres aléatoires.

Ondes circulaires

Si vous lancez une pierre sur la surface paisible d'un étang, des ondes en formes de cercle se propagent à la surface de l'eau. Une prise instantanée fera apparaître en certains endroits des sommets d'onde et en d'autres des points minimum. Or une telle prise instantanée peut être représentée au moyen d'une dispersion de points.

La fonction qui fournit une valeur numérique appropriée pour tout point image possible doit fournir des valeurs égales pour un écart donné par rapport au centre. Pour représenter ce caractère d'onde,

on peut utiliser les fonctions sin ou cos.

```
to ONDE :y
if :y < 0 [stop]
make "x 0
repeat 150 [make "x :x+1 if 10*(1+cos F :x :y) > random
22 [dot se :x :y dot se -:x :y dot -:x -:y dot :x -:y]]
ONDE :y - 2
end
```

Appel par exemple avec: `fs cs ht ONDE 151`

L'opération décrite ci-dessus se déroule dans la liste de la ligne repeat de ONDE. Cette ligne traite tous les points d'une droite horizontale de la fenêtre graphique. Pour représenter une forme d'onde, on prend le cosinus de la distance par rapport à l'origine des coordonnées. Pour ne pas obtenir une valeur de fonction négative, on ajoute d'abord le nombre 1 de sorte qu'on parvient à des valeurs de fonction entre zéro et deux. La multiplication par 10 étend alors les valeurs possibles à une zone de valeurs allant jusqu'à 20. Il est ensuite procédé à la comparaison avec le nombre aléatoire tiré au sort pour la zone de valeurs correspondante sur la taille de laquelle il est également possible de jouer encore un peu.

Pour accélérer l'exécution, on utilise ici la symétrie de la répartition. Le calcul n'est de fait effectué que pour un quart de l'écran, les trois autres quarts étant reproduits par réflexion.

Le programme est simple et il produit un effet graphique grâce à la dispersion aléatoire. Mais comme tous les points possibles doivent être examinés les uns après les autres, le temps de calcul est très important. Nous vous conseillons de ne mettre en oeuvre de telles applications que lorsque vous pouvez laisser l'ordinateur livré à lui-même pendant quelques heures. On peut parfaitement choisir également un plus grand écart entre les différents points et en diminuer ainsi le nombre. Cela se remarquera à peine dans un premier temps du fait de la résolution limitée de l'écran. Cela vaut notamment pour le PCW en ce qui concerne la résolution verticale.

Une fois le dessin terminé, vous pouvez le sauvegarder s'il vous plaît.

```
savepic "ONDEK
```

La fonction de répartition est ici cachée dans la fonction F qui devra être modifiée en conséquence pour obtenir d'autres répartitions. Le résultat de F doit être dans l'exemple considéré un décuple de la distance du point de coordonnées x et y à l'origine des coordonnées. La multiplication par un facteur 10 permet de parvenir à une image intéressante. Vous pouvez faire varier ce facteur à votre guise.

La distance d'un point au point zéro est calculée comme racine carrée de la somme des carrés des coordonnées (théorème de Pythagore). Le fabricant n'a malencontreusement pas jugé bon de fournir dans la version du LOGO dont vous disposez de fonction de calcul de la racine carrée. Cet oubli peut naturellement être comblé par une fonction définie par l'utilisateur mais cela entraîne une perte de temps considérable ici du fait du grand nombre de points à calculer.

L'écart recherché peut cependant également être calculé à l'aide de fonctions trigonométriques qui font partie du vocabulaire de base du DR LOGO.

```
to :F :x :y
if :x=0 [op :y]
if :y=0 [op :x]
op :x/sin arctan :x/:y
end
```

Pour les points situés sur les axes de coordonnées, le calcul à l'aide de la fonction arctan est inutilisable et c'est pour cela que nous devons effectuer des tests dans F. Mais comme ces tests ont encore pour effet de ralentir le calcul, il serait peut-être préférable de traiter les points sur les axes de coordonnées dans des lignes séparées du programme d'appel ONDE. Comme ces points n'auront pas alors besoin de la fonction F, les tests pourront être supprimés de cette procédure.

Pour le dessin présenté par la figure 14.2, nous avons toutefois choisi une méthode d'approximation plus rapide.

```
to F :x :y
make "s :x + :y make "p :x * :y / :s
op 10 * (:s - :p - :p * :p / (:s + :s)
end
```

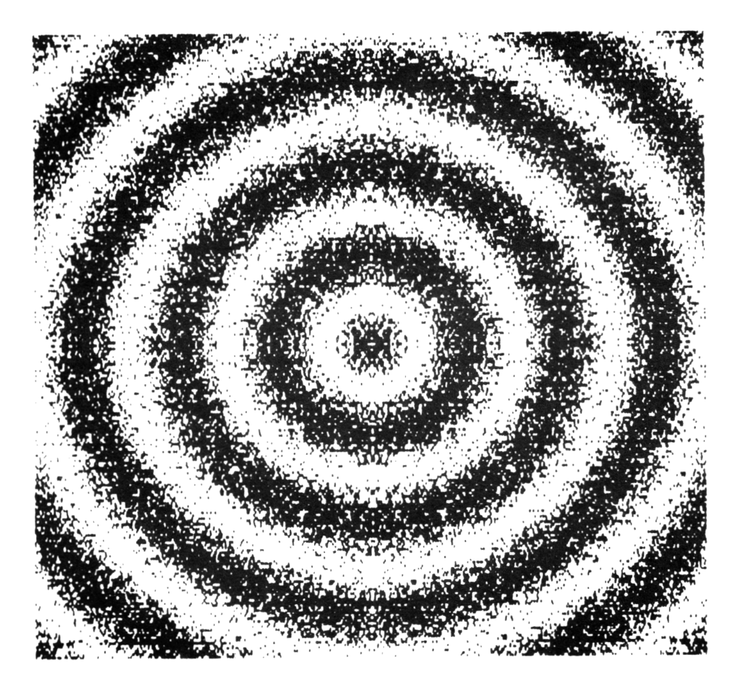


Figure 14.2: Ondes circulaires

Autres répartitions

On peut produire d'autres répartitions en modifiant la procédure F. Il n'y a d'ailleurs, dans ce domaine, aucune limite aux expérimentations possibles. Il peut alors facilement arriver que soit construite une image sans structure clairement identifiable. Il faut alors multiplier la valeur de fonction dans la procédure F par un facteur approprié.

On peut également employer comme outil graphique une extension dans le sens de la verticale. Le LOGO offre pour cela l'instruction setscrunch:

```
setscrunch .2
sf
```

Réponse par exemple: [0 0 WINDOW 0.2]

Cela permet de faire varier le rapport entre les axes. Le réglage standard a la valeur un sur le CPC et 0.46875 sur le PCW. Ces valeurs assurent qu'un déplacement par exemple avec fd 100 produise bien des tracés de même longueur aussi bien horizontalement que verticalement.

L'instruction sf (abréviation de Screen Facts) sort comme dernière valeur la valeur actuelle du facteur définissant le rapport des axes.

Essayez un peu l'exemple suivant qui nécessite un temps de calcul relativement peu élevé.

```
to F :x :y
op 0.05 * :x * :y
end
```

Remarque: dans de telles zones où la valeur de fonction produite par F coïncide, on obtient également des densités de points identiques. C'est pourquoi on obtient des répartitions qui suivent les courbes qui correspondent à l'équation F=constante. Ce sont dans ce cas des hyperboles dont les axes de coordonnées sont les asymptotes.

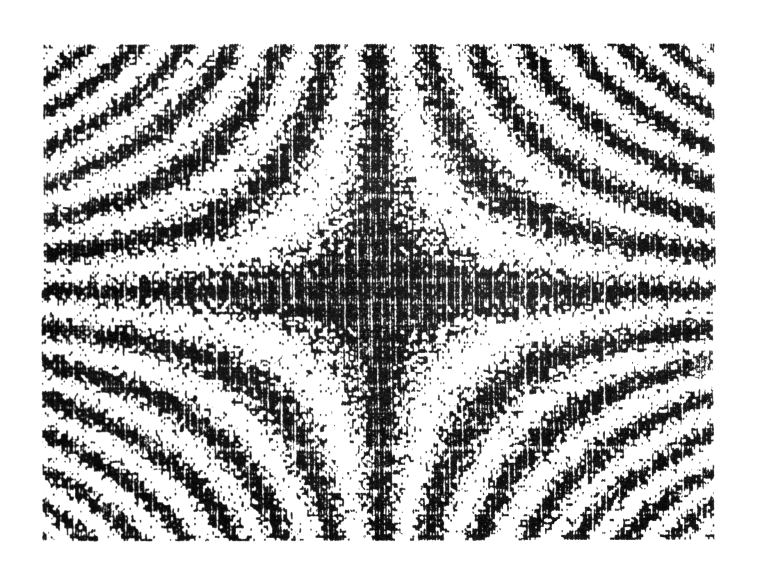


Figure 14.3: Ondes hyperboliques

Le dessin de la figure 14.4 sera obtenu avec la fonction F suivante.

```
to F :x :y
op (2*towards [0 0]) + 700 * LOG :X*:X + :Y*:Y
end

to LOG :x
make "b (word ". :x)
make "t (:b - 1) / (:b + 1)
op :t + 0.8182 * count :x
end

to ONDE :y
if :y < 0 [stop]
make "x -300
```

```
repeat 300 [make "x :x+1 if 10*(1+cos F :x :y) > !
random 22 [dot se :x :y dot se -:x -:y]]
ONDE :y - 2
end
```

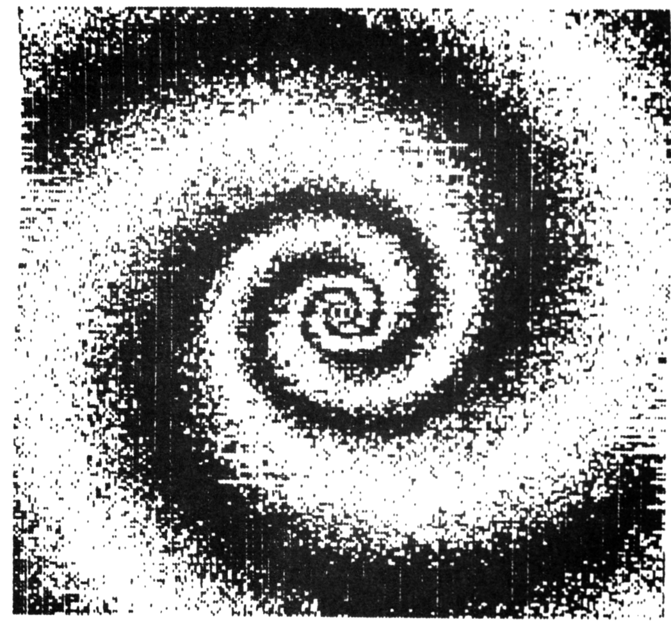


Figure 14.4: Dispersion de points avec une structure en spirale

Le programme principal ONDE a été donné ici une nouvelle fois parce que la figure produite possède une symétrie moindre avec plus qu'une réflexion au point zéro.

La structure de base de la nébuleuse en forme de spirale est une spirale logarithmique. Comme DR LOGO sur les ordinateurs AMSTRAD ne dispose d'aucune fonction logarithmique intégrée, nous avons prévu en remplacement la procédure LOG. Celle-ci fournit cependant une approximation très grossière pour un logarithme dont la base se situe par exemple environ à 18. Cette procédure est parfaitement utilisable pour produire des répartitions de points mais elle n'est certainement pas adaptée pour résoudre des problèmes mathématiques.

La procédure logarithmique de la fonction F est par ailleurs

obtenue essentiellement en décomptant avec count les chiffres décimaux obtenus (l'entrée est ici un nombre naturel).

Hormis le simple plaisir des yeux procuré par des dessins tels que celui que nous vous montrons dans les figures 14.2 à 14.4, de telles répartitions ont aussi des applications en physique où les dispersions aléatoires jouent un rôle réel.

Un exemple bien connu est la probabilité qu'un électron rencontre un obstacle à l'intérieur d'un atome ou d'une molécule. La physique affirme que les électrons, c'est-à-dire les particules chargées d'électricité dans l'enveloppe atomique, ne possèdent pas de situation fixe mais qu'il existe simplement des probabilités déterminées de les rencontrer dans certains endroits. La physique quantique ou la chimie quantique décrivent ces dispersions par des fonctions appropriées. Voilà donc un domaine que vous pourriez illustrer en employant les techniques de programmation que nous venons de décrire.

14.4 Esthétique récursive

Lorsque vous étiez enfant, peut-être aimiez-vous, vous aussi, admirer la beauté des dessins produits par un caléïdoscope. Tout l'attrait de ces dessins provient simplement de la réflexion multiple d'un modèle de base produit par hasard à partir de morceaux de verre. On pourrait donc, de la même manière, employer de manière récursive une étape graphique de base pour produire des modèles attractifs.

Le tétraèdre de la leçon 13 en est un exemple mais nous allons maintenant vous fournir encore quatre autres exemples pour nourrir votre inspiration. Tout d'abord deux exemples dans lesquels l'action de base repose uniquement sur le dessin d'une petite droite. Il est tout à fait étonnant de constater ce qu'on peut faire à partir de cela par de multiples auto-appels et rotations sur des angles droits. Les deux exemples suivants sont tirés de l'ouvrage 'Turtle Geometry' de H. Abelson et A. di Sessa.

```
to C :longueur :niveau
```

```
if :niveau =0 [fd :longueur stop]
C :longueur :niveau - 1 rt 90
C :longueur :niveau - 1 lt 90
end

to DRAGONG :longueur :niveau
if :niveau=0 [fd :longueur stop]
DRAGONG :longueur :niveau - 1 lt 90
DRAGOND :longueur :niveau - 1 end

to DRAGOND :longueur :niveau
IF :niveau=0 [fd :longueur STOP]
DRAGONG :longueur :niveau - 1 rt 90
DRAGOND :longueur :niveau - 1
end
```

Essayez C avec un nombre croissant d'imbrications:

```
fs cs C 5 1 cs C 5 2 ...
```

Les modèles ainsi créés peuvent servir de base au dessin de broches dont on reconnaît nettement la forme caractéristique à partir du niveau 4. Vous constaterez que l'axe de la figure subit une rotation de 45 degrés chaque fois que le nombre de niveaux s'accroît. Avant d'appeler C, vous pouvez alors placer le crayon à dessin dans un emplacement approprié et le mettre dans la bonne direction par une rotation préalable.

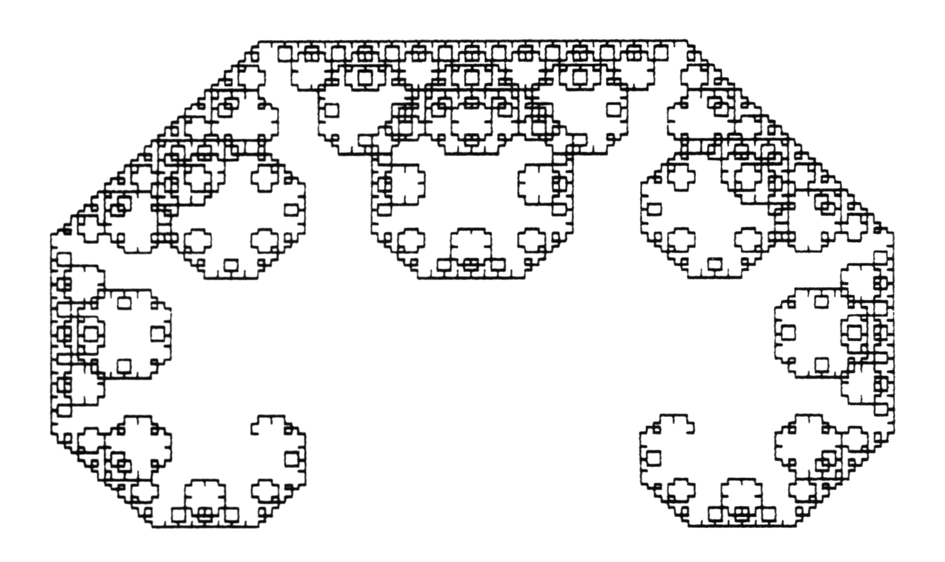


Figure 14.5: Le modèle produit par C au 12ème niveau.

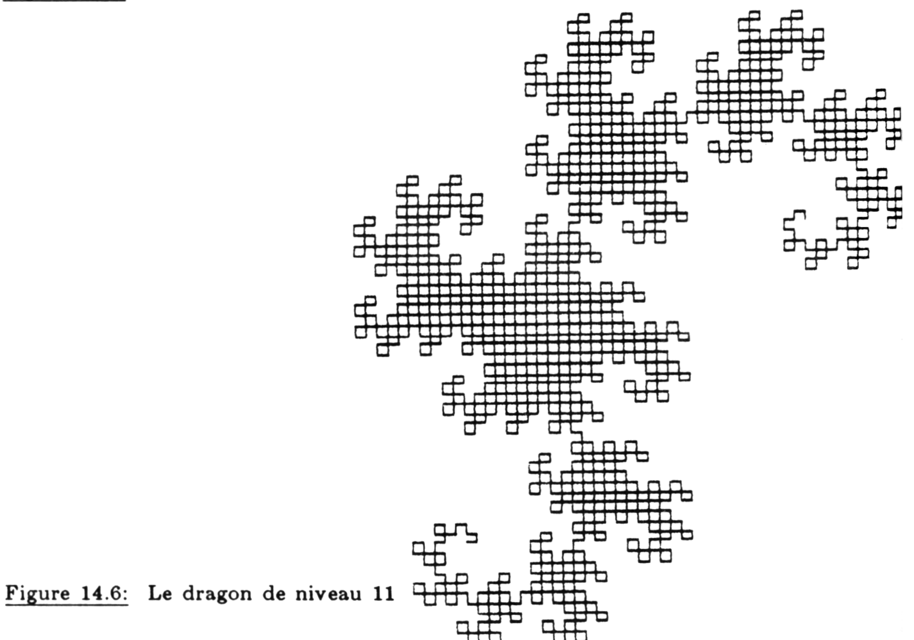


Figure 14.6: Le dragon de niveau 11

Le dessin produit par le deuxième exemple, DRAGONG et DRAGOND, apparaît clairement à partir du niveau 7. Mais essayez aussi:

```
fs cs pu setpos [80 50] pd ht DRAGONG 5 11
```

Pour ces programmes, il est préférable de cacher la tortue avec ht car le triangle vacillant produit sinon une effet plutôt gênant.

Courbes de HILBERT et SIERPINSKI

Les deux prochains exemples sont des courbes qui permettent de peindre des surfaces de manière régulière. La première a été inventée par le mathématicien allemand David Hilbert, la seconde par le polonais Wraclaw Sierpinski.

```
to HILBERT :longueur :niveau :sensrotation
if :niveau =0 [STOP]
if :sensrotation [lt 90] [rt 90]
HILBERT :longueur :niveau - 1 not :sensrotation
fd :longueur
if :sensrotation [rt 90] [lt 90]
HILBERT :longueur :niveau - 1 :sensrotation
fd :longueur
HILBERT :longueur :niveau - 1 :sensrotation
if :sensrotation [rt 90] [lt 90]
fd :longueur
HILBERT :longueur :niveau - 1 not :sensrotation
if :sensrotation [lt 90] [rt 90]
end
```

Les courbes de Hilbert ne sont pas des courbes fermées. Leur modèle de base a la forme d'un U que vous produisez avec:

```
HILBERT 50 1 "TRUE
```

La troisième entrée prescrit un sens de rotation. Avec TRUE, le U sera ouvert vers la droite, avec FALSE il sera ouvert vers la gauche. Au niveau suivant, quatre figures orientées dans les deux directions seront produites et accolées l'une à l'autre par un pas

en avant.

L'emploi de la grandeur sensrotation présente l'avantage de permettre l'utilisation du même programme pour les deux directions possibles alors que dans l'exemple précédent, il nous fallait deux procédures, DRAGONG et DRAGOND. sensrotation correspond à une valeur de vérité, à savoir TRUE ou FALSE. Cette valeur est inversée grâce au mot LOGO not.

```
not "TRUE
```

Réponse: FALSE

Il est particulièrement attrayant de combiner dans un dessin les courbes de Hilbert d'un niveau croissant d'imbrication. Si la longueur du pas est chaque fois divisée par deux et si le point de départ est déplacé vers un point approprié, la courbe de Hilbert du niveau immédiatement supérieur s'entortille autour de celle du niveau précédent. C'est à cela que serviront les programmes-cadres HILBERTK et HILBERTPLOT.

```
to HILBERTPLOT; COURBES DE HILBERT IMBRIQUEES
fs cs HILBERTK 75 -75 1 150 6
end

to HILBERTK :x0 :y0 :i :h :n
if :n=:i [stop]
pu setpos se :x0 :y0 pd
HILBERT :h :i "TRUE
HILBERTK :x0+:H/4 :y0-:h/4 :i+1 :h/2 :n
end
```

HILBERTPLOT vide l'écran graphique et fixe les conditions de départ nécessaires pour que toutes les courbes suivantes, jusqu'au cinquième niveau, soient dessinées sur l'écran graphique avec une taille d'écran de moitié.

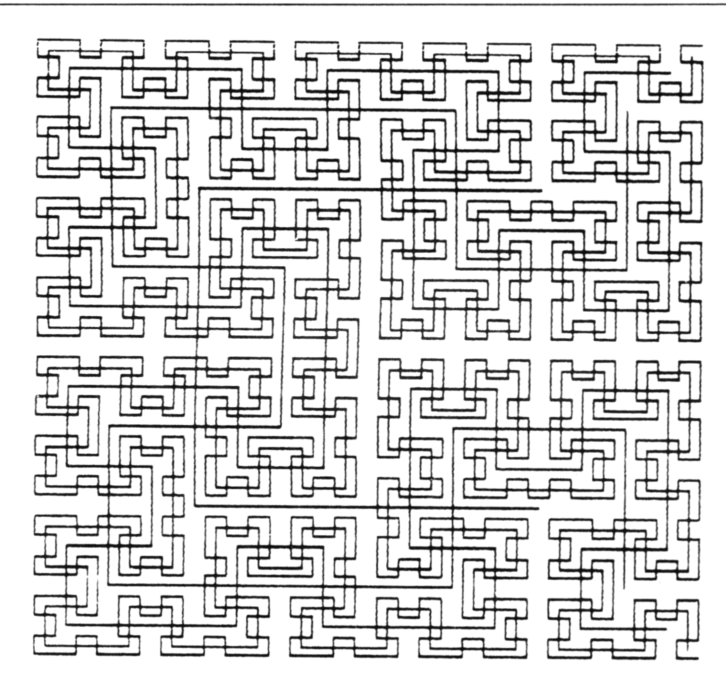


Figure 14.7: Courbes HILBERT superposées

Les courbes de Sierpinski permettent de peindre des surfaces de manière encore plus belle mais la structure récursive est un peu plus difficile à comprendre. Voici d'abord la procédure de base pour produire une courbe de Sierpinski:

```
to A :i;ELEMENT PARTIEL DE SIERPINSKI
if :i=0 [stop]
A :i- 1 rt 135 fd :l lt 45
A :i- 1 fd :l2 lt 180
A :i- 1 rt 135 fd :l lt 45
A :i- 1
end
```

Essayez d'oublier pour le moment l'auto-appel effectué au début de chaque ligne. La procédure A contient trois pas en avant avec les deux distances l et l2. Entre ces pas se produisent chaque fois des rotations de 45 degrés; n'oubliez pas en effet que 180 moins 135 donne aussi 45! Essayez:

```
make "l 50 make "l2 :l*1.4142 A 1
```

Les deux distances utilisées ont le même rapport entre elles que le côté d'un carré par rapport à la diagonale du carré. Observez maintenant l'effet de l'auto-appel au début de chacune des quatre lignes.

```
fs cs A 2 cs A 3 ...
```

La véritable courbe de Sierpinski se compose de quatre éléments ainsi produits par la procédure A, ces éléments étant disposés en angle droit. Un pas en avant avec la direction absolue de 45, 135, -135 puis -45 degrés relie entre eux ces quatre éléments pour constituer une courbe de Sierpinski fermée.

```
repeat 4 [A :i rt 135 fd :l lt 45]
```

i correspond ici au niveau de récursion. Pour vous assurer que les rotations choisies pour A aussi bien que pour cette ligne repeat sont bien celles qui conviennent, le mieux est que vous vous livriez à quelques essais.

Avec ces courbes aussi, il est très attrayant de superposer différents niveaux. C'est ce dont se chargeront les deux programmes-cadres SIERPIN et SIERPINSKI.

```
to SIERPIN :l :l2 :x0 :y0 :i
if :i=:n [stop]
pu setpos se :x0 :y0 pd
repeat 4 [A :i rt 135 fd :l lt 45]
SIERPIN :l/2 :l2/2 :x0-:l2/2 :y0+:l2/4 :i+1
end

to SIERPINSKI :TAILLE :N
fs cs ht
SIERPIN :TAILLE :TAILLE*1.4142 -:TAILLE 2.1*:TAILLE 1
end
```

SIERPINSKI transmet à SIERPIN le point de départ et les deux distances pour le premier niveau. Dans cette dernière procédure, la

courbe est produite dans la ligne repeat au niveau i puis le prochain niveau est mis en oeuvre avec un déplacement approprié du point de départ et avec des distances de moitié. Vous pouvez appeler avec:

```
SIERPINSKI 60 5
```

Ici également, les courbes de niveau plus élevé devraient s'entortiller autour de celles du niveau précédent.

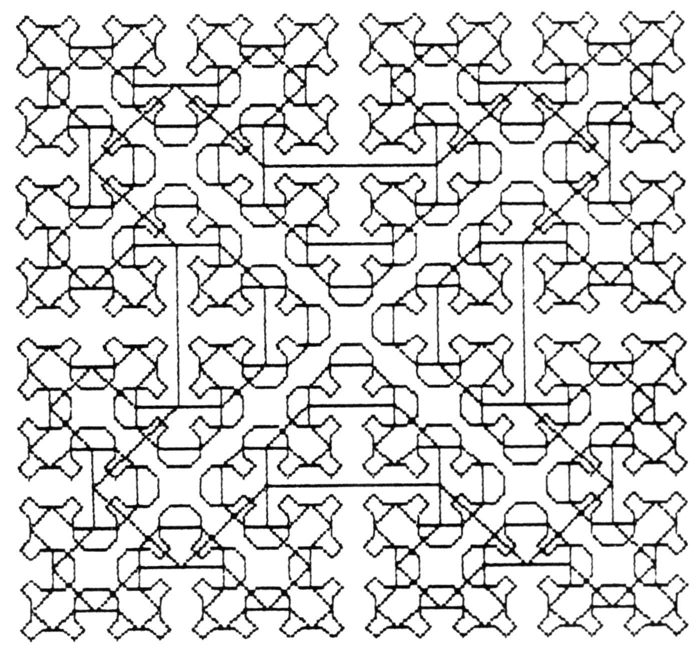


Figure 14.8: Superposition de courbes de SIERPINSKI

14.5 Dessin avec surfaces

Alors qu'il s'agissait à la section 14.2 de construire sur l'écran graphique des dessins à partir de points isolés, les moyens graphiques utilisés à la section précédentes étaient les tracés de traits régulièrement ordonnés. Nous allons maintenant utiliser comme instruments les éléments de surfaces. Les éléments de surfaces sont tout d'abord représentés par des lignes fermés, comme les polygones ou les cercles.

Les surfaces permettent de construire des modèles sur une surface plane, pensez par exemple à la structure des rayons d'une ruche d'abeille. Les surfaces jouent cependant également un rôle essentiel dans la représentation d'objets avec effets de relief dans un dessin. L'effet graphique produit tient non seulement à la représentation du bord de chaque surface mais aussi à l'organisation de l'intérieur de la surface.

Pour faire ressortir l'intérieur de la surface, la surface peut être hachurée, être dotée de points répartis de façon régulière ou aléatoire ou être entièrement représentée dans différents tons de couleur. DR LOGO sur le CPC AMSTRAD n'offre pas d'instructions spécifiques pour organiser les surfaces. Les possesseurs d'un PCW ont un peu plus de chance à cet égard puisque leur version du LOGO prévoit une instruction de remplissage de surfaces.

Remplissage de surfaces sur le PCW AMSTRAD (pas sur le CPC)

Les surfaces peuvent être peintes avec l'instruction fill. Essayez par exemple:

```
to hexagone :l
repeat 6 [fd :l rt 60]
end

hexagone 40 pu rt 60 fd 40 pd fill
```

L'hexagone produit avec hexagone sera rempli entièrement car aucune variation de couleur n'est possible sur le PCW. La ligne d'appel vous permet de déceler les conditions à remplir:

- La tortue doit se trouver à l'intérieur d'un trait de ligne fermé sans être en contact avec le bord. Elle doit pour ainsi dire être libre à l'intérieur de la surface.
- L'état du crayon à dessin doit être pd (ou pe ou px).

Vous pouvez suivre le déroulement de l'opération sur l'écran: des points sont placés les uns au-dessus des autres jusqu'à ce que des

points déjà mis soient atteints. Si vous utilisez fill sans qu'il y ait de bord clos, le remplissage s'étendra sans cesse et il risque de remplir tout l'écran. Notez bien qu'il n'est pas possible d'interrompre fill avant que l'opération ne soit terminée. Il faut donc faire preuve d'une certaine prudence.

Vous pouvez essayer par exemple un modèle simple tel que celui-ci:

```
to hexagones :l
hexagone :l rt 60 pu fd :l pd fill fd :l
hexagone :l rt 60 pu fd 2*:l pd
end
```

Appel: `fs cs repeat 3 [hexagones 30]`

Surfaces hachurées

Les hachures permettent d'organiser des éléments de surface de façons très variées. Si l'on n'est pas soutenu par des éléments de langage intégrés, qui sont en général assez rapide d'exécution, la réalisation de hachures pour n'importe quels types de bords risque de prendre beaucoup de temps car il faut limiter les traits utiliser à l'intérieur de la surface.

Nous prendrons comme exemple une image à bascule bien connue.

```
to PARA :w :n :s
repeat :n [fd :l lt :w fd :s rt :w bk :l lt :w fd :s !
rt :w]
end

to PAR :w :n :s
fd :l lt :w fd :ns rt :w bk :l lt :w bk :ns rt :w
PARA :w :n :s
end
```

Le programme PARA produit des hachures dont la forme extérieure est celle d'un parallélépipède. Essayez par exemple:

```
make "l 60 PARA 60 12 2 PARA 120 4 12
```

Dans le premier cas, les hachures sont tellement étroites qu'on a l'impression d'une surface remplie avec une structure supplémentaire alors que dans le second cas la trace du crayon à dessin peut être nettement identifiée. L'orientation du parallélépipède met en évidence la signification de l'angle constitué par la première entrée de PARA. La procédure PAR dessine maintenant encore en premier lieu une limite supplémentaire.

```
to CHAMP :y
if :y < -180 [stop]
pu setx 320 sety :y seth 0 pd
repeat 8 [PAR 120 2 12 PARA 60 12 2]
pu setx 320 sety :y - :ns - :l seth 0 pd
repeat 8 [PARA 60 12 2 PAR 120 2 12]
CHAMP :y - :ns - :l - :l
end

TO BASCULE
(local "l "ns)
make "l 40 make "ns 48
fs cs ht
CHAMP 150
end
```

Le dessin est produit dans CHAMP. Les deux lignes repeat dessinent chaque fois une série de parallélépipèdes d'orientation et de hachures différents placés les uns à la suite des autres. Les deux séries sont alors accolées l'uneà l'autre de sorte à ce qu'on obtienne le modèle de la figure 14.9.

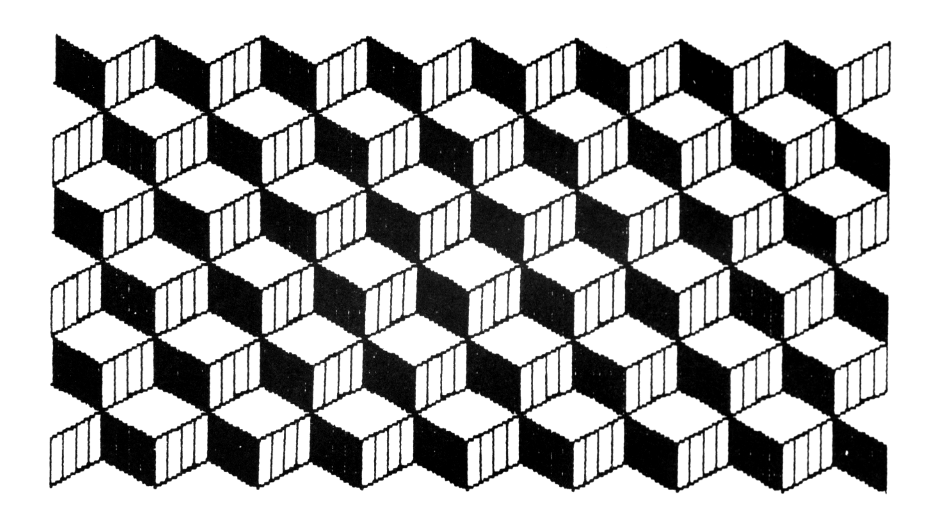


Figure 14.9: Image-bascule à partir de parallélépipèdes hachurés

De telles images peuvent être vues en perspectives de deux manières d'où le terme d'image à bascule. La bascule entre les deux interprétations ne se fait naturellement que dans le cerveau de l'observateur. On a même parfois du mal à se dégager d'une perspective à laquelle on a abouti. Dans le dessin produit par BASCULE, qui n'est évidemment plus qu'un programme-cadre, vous trouvez des surfaces sombres que vous considérerz, dans votre essai d'interprétation dans l'espace, parfois comme allant vers le bas et parfois comme allant vers le haut.

Si vous essayez de vous imaginer que les surfaces à hachures denses sont éclairées par la droite, alors vous verrez les surfaces sombres d'en bas. Si elles vous semblent par contre éclairées de par la gauche alors vous les regarderez d'en haut.

Comme lorsqu'on regarde des dessins on n'interprète souvent en relief qu'une partie de ces dessins en réalité, il est assez facile de tromper notre vision en perspective. Vous avez peut-être déjà vu

des dessins du peintre hollandais Cornelius Escher qui tirent de cette technique une grande partie de leur charme. Ajoutons encore un dessin simple de ce type.

Il s'agira de placer 18 disques de rayon R les uns sur les autres. Les centres correspondront aux coins d'un '18 angles' de côté l. Lorsque des cercles sont disposés les uns sur les autres dans l'espace, on ne peut plus voir du cercle placé en bas qu'une section en forme de faucille.

```
to FAUCILLE :l
rt 110
repeat 4 [rt 10 bk :l]
repeat 24 [fd :l lt 10]
lt 110
repeat 8 [lt 10 bk :l]
repeat 24 [fd :l rt 10]
end
```

Appel: FAUCILLE 10

Le programme de base FAUCILLE forme une faucille à l'aide de deux sections de cercle. Les sections de cercle ont chaque fois un angle d'ouverture de 240 degrés mais elles pivotent. Si FAUCILLE est répété 18 fois, on obtient les 18 faucilles de la figure 14.10.

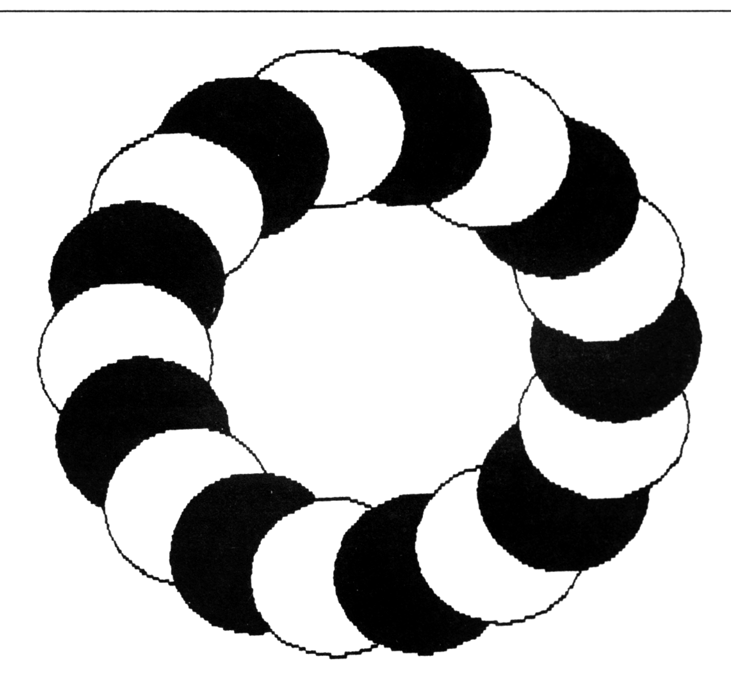


Figure 14.10: Faucilles

Dans la figure 14.10, nous avons rempli une faucille toutes les deux faucilles; cette image a été réalisée sur le PCW AMSTRAD. Le programme correspondant est:

```
to ANNEAUX :l
make "p tf
make "r 5.7296 * :l make "r2 2.8648 * :r
pu fd :r2 + 0.866 * :r pd seth 65
repeat 24 [fd :l rt 10 ]
repeat 9 [FAUCILLE :l]
pu setpos :p seth 0 fd :r2 seth 100;pour le PCW
repeat 9 [pd fill pu fd :r rt 20 fd :r rt 20]; pour le PCW
end
```

Pour remplir les surfaces, la tortue est à nouveau lancée autour du 18 côtés après avoir dessiné les faucilles puisque les coins de ce polygones sont tous situés à l'intérieur de la surface des faucilles. Vous pouvez appeler le programme par exemple avec la

ligne:

```
cs pu ht setpos [-100 100] pd ANNEAUX 5
```

Sur le CPC, les deux lignes portant le commentaire 'pour le PCW' doivent être omises et elles ne seront bien sûr pas exécutées. Vous pouvez cependant essayer peut-être la version suivante qui hachure une faucille sur deux. Le programme devient de ce fait nettement plus compliqué bien que l'effet de hachures obtenu ne soit pas parfait.

```
to FAUCILLE :l
rt 110 repeat 4 [rt 10 bk :l] repeat 8 [fd :l lt 10]
bk :r3 fd :r3 repeat 2 [fd :l lt 10]
repeat 40 [fd :r bk :r rt 5] lt 200;hachurer
fd :l lt 10 fd :r2 bk :r2
repeat 13 [fd :l lt 10]
lt 110
repeat 8 [lt 10 bk :l] repeat 24 [fd :l rt 10]
end

to ANNEAUX :l
make "r 5.7296 * :l make "r2 4.5 * :l make "r3 3.5 * :l
rt 5 repeat 24 [fd :l rt 10]
repeat 9 [FAUCILLE :l]
end
```

Les mouvements de hachures se reconnaissent aux combinaisons de type fd :r bk :r.

14.6 Dessin en trois dimensions

Le dessin en trois dimensions sur ordinateur est un sujet très intéressant mais aussi très complexe qui nécessite certaines bases mathématiques que nous ne pouvons développer ici. C'est pourquoi nous ne ferons qu'une brève incursion dans le domaine de la représentation en trois dimensions. Nous utiliserons les possibilités du graphisme de tortue plutôt que des calculs mathématiques.

Points de fuite

Une des possibilités de représenter la troisième dimension sur une surface de dessin plane est d'employer un point de fuite. Cela consiste à donner l'impression que toutes les droites se coupent parallèlement au troisième axe spatial dans le point de fuite.

La représentation spatiale n'est toutefois pas très facile à construire, même sous la forme élémentaire d'un dé, car les droites placées plus à l'arrière dans l'image doivent se rétrécir mais ce rétrécissement est fonction de la position dans laquelle on se trouve à tout moment par rapport au point de fuite.

Nous prendrons pour base le programme FUITE.

```
to FUITE :cote :direction :butx :buty :fuitex :fuitey
seth towards se :fuitex :fuitey
fd :cote seth :direction
label "debut
fd 1 if PARALLELP :butx :buty :fuitex :fuitey 2 !
[op piece 1 2 tf] [go "debut]
end

to PARALLELP :x1 :y1 :x2 :y2 :tol
make "re remainder (towards se :x1 :y1) - !
(towards se :x2 :y2) 180
if :re < 0 [op -:re > -:tol] [op :re < :tol]
end
```

FUITE dessine d'abord une droite de longueur cote partant de l'emplacement de la tortue en direction du point de fuite. La tortue se dirige ensuite dans la direction indiquée comme valeur en entrée. Après chaque petit pas en avant, on vise sur l'écran le point de fuite ainsi que le point défini par butx, buty. Lorsqu'on relève pour ces deux points des directions parallèles, FUITE est terminé. Les coordonnées du point ainsi atteint en dernier sont renvoyées comme résultat. Le déroulement de l'opération est assez long mais il ne nécessite aucune notion mathématique.

Le mot de test PARALLELP examine si la tortue voit

les deux points (x1, y1) et (x2, y2) dans la même direction ou dans des directions opposées. Du fait du caractère limité de la résolution, l'identité des angles ne peut être examinée qu'avec une certaine marge de tolérance. Plus les points visés sont proches de la tortue et moins la détermination de l'angle sera précise.

```
to DE :cote :fuitex :fuitey
ht
home repeat 4 [fd :cote lt 90]
home fd :cote
make "u FUITE :cote/2 180 0 0 :fuitex :fuitey
home fd :cote
make "o FUITE :cote/2 270 -:cote :cote :fuitex :fuitey
setpos se -:cote :cote
end
```

Le programme est conçu en fonction d'un point de fuite placé sur la droite de l'écran. L'appel peut être effectué par exemple ainsi:

```
cs DE 100 200 200
```

Figure 14.11: Dé avec perspective de point de fuite

Vecteurs et points dans un système de coordonnées en trois dimensions

Il n'est pas facile de se représenter la situation de points, droites et corps divers dans l'espace. Une représentation sur une surface plane est bien sûr aisée à construire à l'aide d'une projection parallèle mais cela prend beaucoup de temps. C'est là que l'ordinateur peut nous rendre de grands services. Ce qui suit n'est qu'un exemple parmi d'autres de représentation d'une perspective.

```
to AXES :n :unite
seth 0 ht pd
repeat 2 [ORIGINE AXE :n :unite rt 90]
seth 60 ORIGINE AXE :n 0.7*:unite
end

to AXE :n :l
fd :n*:l
rt 30 repeat 3 [rt 120 fd 10] lt 30;POINTE DE FLECHE
ORIGINE
repeat :n - 1 [pu fd :l pd rt 90 fd 3 bk 6 fd 3 lt 90]
end

to ORIGINE
pu setpos [-50 -50] pd
end
```

La procédure AXES dessine trois axes de coordonnées dont le second donne l'impression de se perdre vers l'arrière de l'écran. Les droites pointées dans cette direction sont représentées avec un angle de 60 degrés parallèlement les unes aux autres, les longueurs de droites sont uniformément multipliées par un facteur 0.7. L'angle et le facteur peuvent bien sûr être modifiés si l'impression de relief ne paraît pas satisfaisante. Il est nécessaire de raccourcir la troisième direction dans l'espace pour que celle-ci ne paraisse pas surdimensionnée.

On marque des unités sur les axes, la valeur en entrée n indiquant dans AXES la longueur d'axes voulue par rapport à cette unité.

L'origine des coordonnées est fixée par la procédure ORIGINE. Elle peut ainsi être aisément modifiée.

Pour représenter sur l'écran un point indiqué par trois coordonnées, la tortue doit partir de l'origine et effectuer des déplacements en avant parallèlement aux axes de coordonnées. La longueur du mouvement à effectuer est donnée directement par les coordonnées pour les premier et troisième axes mais, pour le second axe, il faut toutefois à nouveau prendre en compte le facteur supplémentaire 0.7. L'unité (en pas graphiques LOGO) est comprise dans ces procédures comme une grandeur globale qui doit donc être fixée au début avec make "unite ... .

Cette méthode à l'aide du graphisme de tortue ne calcule pas les valeurs de coordonnées sur la surface plane du dessin mais les détermine plutôt par la construction d'un dessin.

```
to POINT :point
ORIGINE
op VECTEUR :point
end

to VECTEUR :point
st pu
seth 90 fd :unite * item 1 :point
seth 60 fd .7 * :unite * item 2 :point
seth 0 fd :unite * item 3 :point
ht pd
fd 2 bk 4 fd 2 rt 90 fd 2 bk 4 FD 2;PETITE CROIX
op piece 1 2 tf end
end

to VECTEUR.DIRECTION :vecteur
ORIGINE
op towards POINT :vecteur
end

to DROITE :debut :fin
(local "ak "ek)
make "ak POINT :debut
```

```
make "ek POINT :fin
pd setpos :ak
end
```

Un point ou un vecteur peuvent être fixés dans l'espace à trois dimensions par trois nombres. Ces indications sont réunies par les procédures pour former une liste. POINT et VECTEUR sortent les coordonnées écran des points, sous forme d'une phrase. Elles peuvent être utilisées pour d'autres dessins.

POINT appelle VECTEUR. La seule différence tient au fait que la tortue est d'abord amenée sur l'origine alors qu'avec VECTEUR on commence à partir de la position actuelle. Les coordonnées d'un point se refèrent toujours à l'origine alors qu'un vecteur peut être attaché à n'importe quels points de l'espace.

La procédure DROITE détermine la situation sur l'écran de deux points définis dans l'espace. Elles relie ensuite ces deux points par une droite. Vous pouvez ainsi construire aussi des surfaces et des corps dans l'espace.

La procédure VECTEUR.DIRECTION sort comme résultat la direction sur l'écran, toujours par rapport à la position zéro de la tortue, c'est-à-dire par rapport à la position dirigée vers le haut, de sorte que vous pouvez par exemple dessiner des droites dont vous indiquez la direction.

Lorsqu'on détermine des points ou des vecteurs sur l'écran, les mouvements de la tortue sont visibles et le crayon à dessin est relevé. Le point est marqué par une petite croix et la tortue est ensuite cachée pour qu'on puisse voir la position exacte du point.

Exemples:

```
make "unite 30 DROITE [2 2 1] [-2 2 -2]
```

Une droite sera dessinée.

```
POINT [1 -1 2] VECTEUR [0 1 1]
```

Deux points seront représentés. Le second point est donné par addition de vecteurs. On peut ainsi examiner l'addition de vecteurs dans l'espace à trois dimensions. Le second point qui est obtenu comme point final de l'addition du vecteur doit se confondre, d'après les règles du calcul vectoriel avec:

```
POINT [1 0 3]
```

Voici maintenant une procédure qui nous servira d'exemple pour la représentation d'un corps dans l'espace:

```
to TETRAEDRE :g
(local "a "b "c "d)
make "a (se 0 0 1.5 * :g)
make "b (se :g :g 0 )
make "c (se -:g 0 0 )
make "d (se 0.5 * :g -:g 0 )
setpc 2 DS :a :c :b 0.05
setpc 3 DS :b :d :c 0.05
setpc 1 DS :a :c :d 0.02
end
```

Pour représenter les surfaces limites, on utilise ici dans TETRAEDRE la définition des couleurs avec l'instruction setpc, ce qui n'aura bien sûr un plein effet que sur un CPC avec moniteur couleur.

Dans la procédure DS, on dessine chaque fois l'un des côtés délimitant le triangle. La formulation d'une procédure spécifique pour cette tâche a aussi pour but de permettre par la suite d'améliorer encore la représentation des superficies. La dernière entrée n'a aucune signification pour le moment.

Triangles hachurés

Nous allons hachurer les côtés du triangle. Comme cette opération est intéressante indépendamment de la représentation d'un tétraèdre, nous allons vous en présenter ici une solution séparée.

```
to HACHURE :f :ax :ay :bx :by :cx :cy
if :f > 0.99 [stop]
pu setpos se :f * :ax + :cx :f * :ay + :cy
pd setpos se :f * :bx + :cx :f * :by + :cy
HACHURE :f + :df :ax :ay :bx :by :cx :cy
end

to DS :a :b :c :df
pu setpos :a pd setpos :b setpos :c setpos :a
HACHURE :df (first :a) - (first :c) (last :a) - (last !
:c) (first :b) - (first :c) (last :b) - (last :c) first !
:c last :c
end
```

Une surface est hachurée à l'aide de tout un groupe de droites parallèles. Avec des triangles, le plus simple est de tracer des parallèles à l'un des trois côtés. Les points initial et final sont déterminés dans les procédure HACHURE et les droites sont ensuite tracées avec setpos.

Essayez cette procédure dans un premier temps indépendamment de la représentation du tétraèdre, en l'appliquant à un triangle simple:

```
DS [0 0] [100 100] [-100 100] 0.1
```

La dernière entrée détermine le nombre et l'écartement des parallèles formant les hachures. Le mieux est que vous en étudiiez l'effet par tâtonnement.

Sur le PCW AMSTRAD, vous pouvez utiliser également l'instruction fill pour remplir une surface de triangle. Les trois dernières lignes de TETRAEDRE peuvent alors être remplacées par exemple par les suivantes:

```
DS :a :c :d 1
pu setpos :c seth 90 fd 10 pd fill pu
DS :a :c :b 0.1
DS :b :d :c 0.1
```

C'est ainsi que nous avons réaliseé le dessin de la figure 14.11.

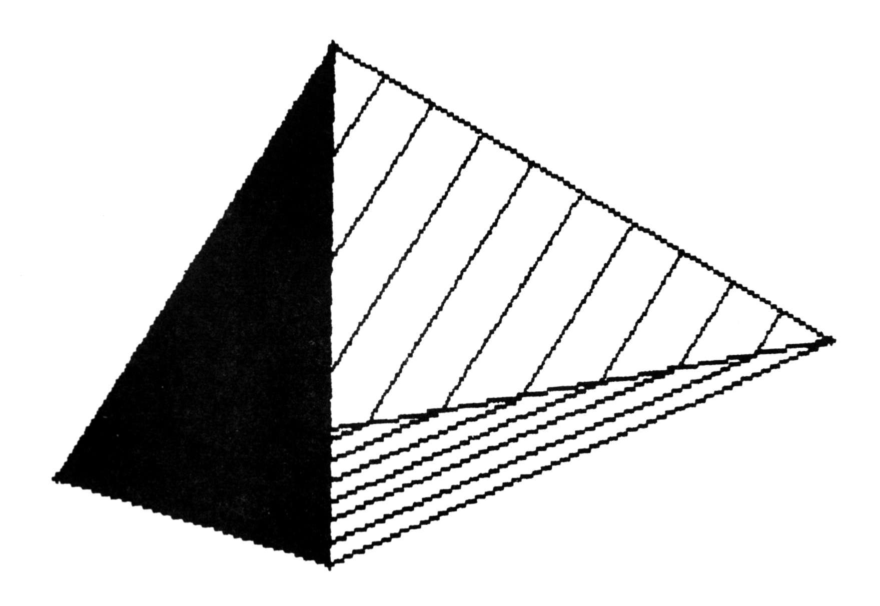


Figure 14.12: Représentation d'un tétraèdre

Leçon 15: Travail approfondi avec les listes

Nouveaux éléments de vocabulaire:
or, and

Programmes:
REMPLACER, CONSERVER, INSERER, ELIMINER, CHANGEPARAGRAPHE, SALUTATION (avec programmes utilitaires, programme de dialogue avec modèles pour expressions du langage)

Sous LOGO, les suites de caractères sans espace de séparation s'appellent des mots, quelle que soit leur signification intrinsèque. A partir de mots, on peut former des phrases en séparant les mots par des espaces et en les réunissant comme phrase avec des crochets. Une phrase est alors appelée liste. Les listes sont composées d'éléments qui peuvent être non seulement des mots mais aussi à leur tour des listes.

15.1 Remplacement d'éléments de listes

Pour manipuler des phrases, il faut pouvoir disposer d'opérations permettant de remplacer différents mots, donc différents éléments d'une liste, par d'autres. Il faut pour cela que les différents éléments puissent être isolés, examinés, éventuellement remplacés par d'autres mots et ensuite à nouveau insérés dans la liste. La suppression et l'insertion d'éléments devant ou après certains éléments sont des manipulations qui se déroulent suivant un schéma semblable.

Les éléments d'une liste peuvent être isolés avec la procédure LOGO first ou bien, en commençant par la fin, avec last. Un mot peut ensuite être à nouveau inséré avec fput:

```
fput (OPERATION first :LISTE) bf :LISTE
```

ou aussi avec se

```
se (OPERATION first :LISTE) bf :LISTE
```

Suivant l'effet de la procédure OPERATION, cette ligne manipule le premier élément. Si le mot doit par exemple être renversé par OPERATION, OPERATION pourrait être remplacée par la procédure RENVERSER que nous avions développée à la leçon 9.

```
TO RENVERSER :mot
if emptyp :mot [op :mot]
op word RENVERSER bf :MOT first :MOT
end
```

Exemple:

```
make "PHRASE [MANIPULATION DE PHRASES]
se RENVERSER first :PHRASE bf :PHRASE
```

Réponse: [NOITALUPINAM DE PHRASES]

Pour manipuler chaque mot de la phrase, la modification du premier mot doit être à nouveau appliquée au reste de la phrase. Le reste de la phrase comprend un mot de moins. Nous pouvons maintenant formuler une solution récursive.

```
to REMPLACER :PHRASE
if emptyp :PHRASE [op :PHRASE]
op se CHANGER first :PHRASE REMPLACER bf :PHRASE
end

to CHANGER :MOT
op RENVERSER :MOT
end
```

Appel: REMPLACER [DEMAIN LE SOLEIL BRILLERA]
Réponse: [NIAMED EL LIELOS ARELLIRB]

Appel : REMPLACER REMPLACER [DEMAIN LE SOLEIL BRILLERA]
Réponse : [DEMAIN LE SOLEIL BRILLERA]

La procédure CHANGER, dans REMPLACER, aurait bien sûr pu aussi s'appeler RENVERSER. Le schéma employé dans la procédure REMPLACER peut être étendu à toutes les manipulations de phrases. C'est pourquoi nous avons conservé à cette procédure un caractère plus général.

La manipulation se déroule dans REMPLACER selon un schéma récursif typique: on retranche chaque fois le premier élément d'une liste et on procède de même avec le reste de la liste. Ce n'est qu'une fois que le dernier élément de la liste a été atteint que la liste modifiée est à nouveau reconstituée, étape par étape, avec se. Les mots LOGO first, bf, last, bl, fput, lput se prêtent particulièrement bien à ce type d'opérations.

La procédure décrite devrait fonctionner sur toutes les versions du LOGO. DR LOGO, qui est une version particulièrement puissante du LOGO, vous offre cependant encore d'autres moyens qui vous permettent un accès non-récursif aux éléments d'une liste.

- item :N :PHRASE fournit le Nième élément d'une liste.

- piece :I :N :PHRASE fournit la section de liste de l'élément I à l'élément N.

Remplacement et insertion

item permet d'accéder directement à un elément de liste mais il n'existe pas de mot LOGO qui permette à l'inverse de remplacer ou de modifier un élément de liste. C'est pourquoi nous allons formuler un mot LOGO REPLACE à cet effet, mot auquel nous pourrons ensuite avoir recours.

```
to REPLACE :i :L :W
if :i > 1 [make "W lput :W piece 1 :i - 1 :L]
if :i < count :L [op se :W piece :i+1 count :L :L] [op :w]
end
```

```
to INSERER :i :L :W
if :i > 1 [make "W lput :W piece 1 :i - 1 :L]
if :i > count :L [op :W] [op se :W piece :i count :L :L]
end
```

REPLACE permet de remplacer dans la liste L le i-ième élément par la valeur d W. Au contraire de la méthode utilisée dans le programme REMPLACER donné plus haut, qui consistait à décomposer et à remplacer de façon récursive, on détache ici, avec piece, les parties de la liste placées avant ou après l'élément sélectionné. La liste est ensuite recomposée avec l'entrée W à l'aide de se.

La procédure INSERER ne se différencie que peu, dans la troisième ligne, de REPLACE. L'élément de liste sélectionné avec i est ici repris dans la partie postérieure de la liste. INSERER permet donc d'insérer la troisième entrée dans la position i de la liste L.

Dans ces deux procédures, la liste L peut être aussi bien une phrase simple qu'une liste composée.

REPLACE 2 [a x c d e] "b
Résultat: [a b c d e]

REPLACE 3 [[Centrale hydraulique] [Mines de charbon] [Energie atomique]] [Energie solaire]
Résultat: [[Centrale hydraulique] [Mines de charbon] [Energie solaire]]

INSERER 2 [Jean Rousseau] "Jacques
Résultat: [Jean Jacques Rousseau]

Il arrive aussi qu'on veuille simplement éliminer un élément de liste. Cela peut bien sûr être fait de façon très semblable au remplacement.

```
to ELIMINER :i :L
if :i =1 [op bf :L]
if :i = count :L [op bl :L] [op se piece 1 :i-1 :L piece !
:i+1 count :L :L]
end
```

```
ELIMINER 3 [Rouge Vert Bleu]
```

Résultat: [Rouge Vert]

Suppression d'une partie des procédures présentes

Il arrive souvent que se soient accumulées dans la mémoire de travail de nombreuses procédures dont on n'a plus besoin. Il peut s'avérer assez pénible de devoir les supprimer les unes après les autres et il peut aussi facilement arriver qu'on en supprime ainsi certaines par erreur. La procédure ELIMINER peut se révéler fort utile pour résoudre ce problème.

```
to CONSERVER :l
(pr [Faut-il supprimer toutes les procedures a part] :l !
[o/n]
if lc first rq = "n [stop]
er RESTE se :l [ELIMINER CONSERVER RESTE] glist ".DEF
end

to RESTE :l :p
if emptyp :l [op :p]
if memberp first :l :p [op RESTE bf :l ELIMINER where :p]
[op RESTE bf :l :p]
end
```

Après une question de sécurité, toutes les procédures sauf celles énumérées dans la liste en entrée et sauf celles dont nous avons besoin ici, RESTE, ELIMINER, CONSERVER, seront supprimées. Si vous voulez supprimer également ces dernières, la quatrième ligne de CONSERVER devra se présenter ainsi:

```
er RESTE :l glist ".DEF
```

L'instruction glist ".DEF (n'oubliez pas les majuscules pour DEF) produit une liste de toutes les procédures définies par l'utilisateur actuellement présentes en mémoire.

Remplacement d'un mot recherché par un autre mot

Tout le monde connaît maintenant les lettres personnalisées produites en grande masse à l'aide d'ordinateurs. Le destinataire peut avoir l'impression qu'elles lui sont adressées personnellement parce que le nom du destinataire s'insère automatiquement dans des emplacements modifiables du texte de départ. Il suffit au programme de rechercher ces emplacements et des les remplacer par le nom qui convient, nom qui est bien sûr tiré d'un fichier qui peut compter plusieurs milliers de noms. Nous allons tout d'abord examiner une phrase unique sans marques de phrase.

Nous allons remplacer dans une phrase un mot déterminé par un autre. En employant REPLACE, nous arrivons à la solution suivante:

```
to ECHANGER :phrase :ancien :nouveau
if memberp :ancien :phrase !
[op ECHANGER REPLACE where :phrase :nouveau :ancien !
:nouveau] [op :phrase]
end
```

Cette formulation utilise la propriété du mot de test memberp. En effet, DR LOGO doit rechercher si le mot entré sous le nom ancien figure dans la liste phrase. Avec memberp, la réponse est bien sûr TRUE ou FALSE. Mais ensuite, on peut aussi savoir, grâce au mot LOGO:

```
where
```

en quelle position le mot a été trouvé. Cette indication sera ensuite nécessaire lors de l'appel de REPLACE pour remplacer le mot trouvé par le mot appelé nouveau. Comme where n'indique que la première position dans laquelle le mot a été trouvé, l'interrogation doit être répétée jusqu'à ce que le mot recherché ne fasse plus partie de la liste.

Le mot recherché n'a souvent qu'une fonction de variable mise à la place d'un verbe ou d'un nom, etc... On peut donc tout à fait utiliser comme variable simplement un caractère spécial tel que & par exemple. Si l'on utilise pour cela des caractères spéciaux qui

ont une signification particulière pour le LOGO, par exemple *, +, etc..., ceux-ci doivent être précédés d'un Back Slash qui peut être produit également avec CONTROL+Q (sur le PCW, ALT+Q ou touche f3).

```
ECHANGER [Jean Verbe Sandrine] "Verbe "aime
```

Réponse: [Jean aime Sandrine]

```
ECHANGER [Nous aimons &] "& "chanter
```

Réponse: [Nous aimons chanter]

```
ECHANGER [&, vous avez gagne!] "& "Monsieur\ Durand
```

```
ECHANGER [Paul & Monique] "& "est\ completement\ fou\ de
```

La solution avec REPLACE présente l'inconvénient suivant: un élément de liste doit toujours être remplacé par exactement un nouvel élément. Si un mot doit être remplacé par plusieurs mots, on peut former une liste à partir de ces mots puis remplacer ensuite l'ancien mot par la nouvelle liste. Mais dans ce cas, le résultat ne sera pas une phrase mais une liste imbriquée, comme le montrent les crochets entourant le résultat.

Dans les deux derniers exemples, un mot unique, au sens de la syntaxe du LOGO, est formé à partir de plusieurs mots qui sont séparés entre eux par des espaces précédés chaque fois du caractère Back Slash. On pourrait cependant généraliser le programme ECHANGER à l'aide de la procédure INSERER.

Si plusieurs mots doivent être remplacés dans la phrase, la boucle de ECHANGER devra être parcourue pour chaque mot. Dans l'extension du programme que voici, on attend chaque fois même des phrases, sous les noms ancien et nouveau, qui sont alors décomposées avec first et bf.

```
to ACHANGE :phrase :ancien :nouveau
if emptyp :ancien [op :phrase]]
op ACHANGE ECHANGER :phrase first :ancien first :nouveau !
bf :ancien bf :nouveau
end
```

Exemple:

```
ACHANGE [& brillera $] [& $] [Demain le\ soleil]
```

Réponse:

```
Demain brillera le soleil
```

15.2 Remplacement dans des textes

Un long texte doit être de préférence organisé en chapitres, sections et phrases. Pour la syntaxe du LOGO, cette organisation doit être effectuée au moyen des crochets car les signes de ponctuation font partie du contenu même du texte. Pour remplacer un mot dans un paragraphe, on parcourt chaque phrase du texte et on fait l'échange dans chaque phrase.

```
to CHANGEPARAGRAPHE :liste :ancien :nouveau
if emptyp :liste [op []]
op fput ACHANGE first :liste :ancien :nouveau !
CHANGEPARAGRAPHE bf :liste :ancien :nouveau
end

to TEXTE1
op [[& est un langage de programmation polyvalent.]
[Meme les enfants peuvent deja apprendre & et ecrire !
leurs propres programmes en &.]
[mais & convient aussi pour des problemes plus complexes !
car les recursions et le traitement de listes sont !
possibles dans les procedures &.]
[les programmes peuvent etre examines et modifies, !
sous &, par des programmes.]
[le & offre ainsi les conditions ideales pour les !
```

```
end
```

Nous avons ici programmé un échantillon de texte dans la procédure TEXTE1. Le texte doit être entré comme une seule ligne ce qui, étant donné sa longueur, n'est possible qu'avec l'éditeur LOGO, c'est-à-dire après avoir entré ed "TEXTE1. Essayez maintenant:

```
CHANGEPARAGRAPHE TEXTE1 [& &,] [LOGO LOGO,]
```

Vous n'êtes peut-être pas de cet avis et vous seriez plutôt pour:

```
CHANGEPARAGRAPHE TEXTE1 [LOGO LOGO,] [PASCAL PASCAL,]
```

15.3 Recherche dans le contexte: programmes de dialogue intelligents

De nombreux programmes d'application, par exemple les jeux électroniques ou les programmes de gestion, conduisent au début un dialogue avec l'utilisateur. On demande ainsi des renseignements tels que le nom ou la date pour pouvoir réutiliser ces indications dans des phases ultérieures du programme. L'utilisateur a en général peu de liberté dans ses réponses et il est conduit par des instructions précises ou par des masques écran. Si la forme de l'entrée ne convient pas, celle-ci doit être répétée et il peut même arriver que certaines entrées incorrectes se traduisent par des erreurs fatales.

Les dialogues entre êtres humains se déroulent de manière très différente. La personne interpelée peut réagir à une question par des réponses formulées de façons très variées et cela ne fait pas obstacle à la compréhension. Nous sommes habitués à disposer d'un éventail très étendu d'expressions du langage et nous avons appris à évaluer la signification des mots et concepts d'après leur contexte. En cas d'ambiguïté, il suffit d'ailleurs de demander une confirmation. Dans ce cas, il est bien sûr souhaitable que la demande de confirmation ne soit pas simplement une répétition de la première question.

Il serait certainement souhaitable que les dialogues dans les

programmes informatiques se conforment à ce modèle. Nous allons maintenant nous livrer à un petit exercice préparatoire dans ce sens: il s'agira d'un programme qui pourra retrouver dans un texte des mots qui ne peuvent apparaître que dans des contextes déterminés.

Un dialogue doit commencer par la demande du nom:

Quel est ton nom?

Un programme rudimentaire n'accepterait comme réponse que le nom lui-même, c'est-à-dire exactement un mot. Un programme intelligent doit par contre être à même de traiter correctement également des phrases du type suivant:

Je m'appelle Linda.
Mon nom est Antoine.
Alain est mon nom
Je crois que je m'appelle Charles.
Oui, mon nom est Alex.

Les réponses sont différentes mais les noms apparaissent chaque fois dans des environnements de mots caractéristiques. Nous allons donc développer une procédure qui soit capable de reconnaître un tel environnement de mots dans une phrase entrée. Le mot devant être interprété comme le nom devra alors être retrouvé et mis à disposition pour une utilisation ultérieure.

Les environnements de mots possibles serviront de modèles pour l'examen de la phrase entrée. Seules certaines parties d'une phrase doivent être conformes au modèle alors que les parties de phrase non-pertinentes devront être représentées par un caractère joker, par exemple par un signe $. Le mot devant être interprété comme le nom sera représenté par un autre signe joker, par exemple par le caractère &. Voici par conséquent des modèles possibles:

```
Je m'appelle &, Je suis $ &
```

Ce modèle admettrait des réponses du type: Oui, je m'appelle Jules, Je pense que je m'appelle Alice, Qui es-tu, je suis ta Julie? Le

programme que nous voulons développer devra parcourir la phrase entrée avec ce modèle. Les mots obligatoires devront être conformes alors que par contre le caractère joker acceptera n'importe quel mot. Si toutes les comparaisons confirment la conformité au modèle, le mot trouvé dans l'emplacement correspondant au caractère joker & sera accepté comme nom.

Un tel programme peut bien sûr aboutir à des résultats absurdes car les noms sont simplement identifiés d'après leur position dans la phrase. On pourrait donc réfléchir ici à un examen du caractère plausible de la réponse. Cet examen aurait pour fonction d'exclure des erreurs prévisibles.

Lors de la comparaison au modèle, on doit d'abord concentrer les recherches sur le premier mot obligatoire. On examinera la phrase avec memberp pour savoir si elle le contient. Si le mot est effectivement rencontré, le mot LOGO where nous permettra de déterminer sa position dans la phrase.

```
to CHERCHER :phrase :modele
if op first :modele = "$ first :modele = "& [op CHERCHER !
bf :phrase bf :modele]
if memberp first :modele :phrase [op where] [op 0]
end
```

La seconde ligne, dans CHERCHER, sert à passer par dessus les caractères joker. Lorsque le résultat n'est pas zéro, c'est-à-dire lorsque le premier mot obligatoire a été trouvé, le reste doit être comparé. C'est à cela que sert la procédure COMPARERP.

```
to COMPARERP :phrase :modele
if emptyp :modele [op "TRUE]
if emptyp :phrase [op "FALSE]
local "mot make "mot first :modele
if (or :mot = first :phrase :mot <> "$ :mot = "&) !
[op COMPARERP bf :phrase bf :modele] [op "FALSE]
end
```

COMPARERP est un mot de test qui fournit donc TRUE ou FALSE comme résultat. Faux indique que la phrase a été terminée avant le

modèle et que le premier mot du modèle n'était ni un caractère joker ni un mot identique au premier mot de la phrase. Vrai indique que le modèle a été parcouru sans que soit remplie une des conditions pour faux.

Dans l'avant-avant-dernière ligne, trois conditions sont reliées logiquement entre elles avec or.

CHERCHER et COMPARERP seront appelées dans le programme TROUVER. L'entrée pour TROUVER sera une liste de modèles qui seront parcourus de manière récursive.

```
to TROUVER :modeles
if emptyp :modeles [op "]
(local "modele "st "s)
make "modele first :modeles
make "st CHERCHER :phrase :modele
if (or not memberp "& :modele :st=0 :st=:sl)!
[op TROUVER bf :modeles]
make "s where
if COMPARERP piece :st+1 :sl :phrase bf :modele !
[op item :st+:s - 1 :phrase] [op TROUVER bf :modeles]
end
```

Les grandeurs st ou s conservent respectivement la position dans laquelle a été trouvé dans la phrase le premier mot obligatoire et la position dans laquelle le caractère joker &, représentant le nom recherché, figure dans le modèle.

Notez que l'auto-appel doit être effectué avec op car TROUVER est une procédure avec résultat. Si un mot est identifié comme nom, alors c'est le résultat, sinon un mot vide sera renvoyé.

Le mot de test COMPARERP est appelé dans la septième ligne.

Il nous faut encore prendre en compte un cas particulier lors de l'examen de la phrase. La réponse à la question 'Quel est ton nom?' peut aussi ne se composer que du nom. C'est pourquoi TROUVER sera appelé à partir d'un petit programme-cadre appelé TROUVE. Pour que nous puissions maintenant conduire un dialogue complet, il nous

faut encore formuler un programme de contrôle : SALUTATION.

```
to SALUTATION
label "deb pr [Quel est ton nom?]
make "nom TROUVE CONVERTIR rl []
if emptyp :nom !
[pr [Excuse-moi. Je n'ai pas compris le nom.] !
go "deb]
pr (se [Content de parler avec toi] :nom "! )
end

to TROUVE :phrase
local "sl make "sl count :phrase
if :sl=1 [op first :phrase] [op TROUVER MODELES]
end
```

Dans le programme cadre SALUTATION, la réponse à la question "Quel est ton nom?", à laquelle il est répondu avec rl, est tout d'abord transmise à la procédure CONVERTIR. La première tâche de CONVERTIR est d'éliminer les signes de ponctuation de la réponse. Les signes de ponctuation sont généralement accolés au mots sans intervalle. C'est pourquoi ils sont considérés par le LOGO comme faisant partie des mots. Les noms avec un signe de ponctuation éventuellement accolé sont donc tout d'abord identifiés. Ensuite ce n'est pas la même chose, pour la comparaison avec le modèle, que "mon nom" figure par exemple au milieu de la phrase, devant un membre de phrase ou à la fin de la phrase.

```
to CONVERTIR :l :a
if emptyp :l [op :a]
local "w make "w lc first :l if ascii last :w < 65 make !
"w bl :w
op CONVERTIR bf :l se :a :w
end
```

Un autre problème tient au fait que DR LOGO pour les ordinateurs AMSTRAD dispose des deux écritures en majuscules et minuscules. C'est pourquoi les mots de la réponse fournie sont systématiquement convertis en minuscules dans la procédure CONVERTIR. On n'aura pas ainsi à tenir compte dans le modèle du fait que les mots de même

signification peuvent être écrits différemment selon qu'ils figurent ou non en début de phrase. Il faudra cependant que le nom trouvé soit à nouveau converti en majuscules à la fin.

A la fin de TROUVE, le programme TROUVER est appelé avec MODELES en entrée. Comme il n'y a pas de double point initial, il s'agit d'un nom de procédure. On a donc besoin encore d'une autre procédure qui sorte une liste de modèles. Lorsqu'on voudra augmenter le réservoir de modèles, il suffira de modifier cette seule procédure.

```
to MODELES
op [[& est $ nom] [je m'appelle &] [mon nom est &]]
end
```

Le programme sera alors à même de réagir correctement à des réponses aussi variées que les suivantes:

```
Jean
Comment, mon nom est Alex
Antoine est mon nom, je vous prie
Je m'appelle Monique pour autant que je sache
Mon Dieu, je m'appelle Albert bien sûr
Salut, Piedoie est mon nom
Ma oueille, mon nom est Xavier, qu'est-ce ti croue?
Bonjour, je m'appelle Maxime
```

Les noms Jean, Alex, Antoine, Monique, etc... seront identifiés comme tels.

Le programme que nous vous avons proposé est loin d'être parfait! Si on entre par exemple deux mots pour le nom, par exemple un prénom et un nom de famille, ce qui, après tout, n'a rien d'extraordinaire, seul le premier mot sera identifié comme nom. Il n'est pas non plus possible d'utiliser des modèles qui se chevauchent comme par exemple je suis & et je suis Monsieur &. Si notre version ne vous satisfait pas, il ne vous reste plus qu'à vous en saisir pour doter ce programme de dialogue de plus d'intelligence.

Le LOGO et My Fair Lady

Le programme Eliza, développé par J. Weizenbaum, est un célèbre programme de dialogue. Il est ainsi appelé à cause du personnage principal de la comédie Pygmalion, de Shaw, qui a servi de base à My Fair Lady. Ce programme était réputé être capable de conduire un dialogue apparemment sensé et compréhensible entre le médecin et son patient, l'ordinateur occupant la position du médecin.

Les programmes de ce type exploitent le fait qu'on utilise des formules et des expressions toutes faites dans certaines situations déterminées. Le travail du thérapeute tend d'abord à faire s'exprimer son interlocuteur tout en ne s'impliquant pas lui-même. Les réactions du médecin dans la première phase du dialogue laissent à peine entrevoir s'il a réllement compris les réponses du patient.

C'est pourquoi il n'est pas très difficile d'écrire des programmes simples pour le début d'un tel dialogue. Les réactions du programme aux réponses du patient peuvent être constituées au départ par des formules standard telles que 'Continuez', 'Depuis combien de temps?', 'Avez-vous d'autres douleurs?'. Elles peuvent également être constituées par des répétitions des affirmations du patient sous une autre forme. Pour que le déroulement des réponses puisse varier, on peut prévoir que le programme puise, à l'aide de nombres aléatoires, dans un répertoire de phrases ou de parties de phrases. Il est déjà plus difficile de répéter les affirmations entrées par le patient d'une façon qui soit correcte sur le plan grammatical. En effet, il faut alors modifier dans la phrase au moins la personne et donc aussi modifier la forme du verbe pour l'accorder avec le changement de personne.

Le programme SALUTATION représentait déjà un progrès dans l'intelligence du programme puisque la signification des mots y était analysée en fonction du contexte, c'est-à-dire de l'environnement du mot. Pour réaliser un programme intelligent de dialogue avec un patient, vous pourriez essayer de vous demander quelles suites de mots caractéristiques peuvent se rencontrer dans les expressions employées par le patient. Si vous parvenez ensuite à développer des modèles qui vous permettent de déterminer la

signification de certains mots dans des réponses de ce type, alors vous pourrez obtenir des réactions du programme encore plus intelligentes.

Une question précise telle que 'Quel est ton nom?' restreint déjà considérablement le champ des réponses qu'on peut attendre d'un interlocuteur sérieux. C'est pourquoi on peut fort bien imaginer qu'un programme simulant la position du médecin oriente fortement les réponses de l'interlocuteur.

Il serait toutefois insensé de vouloir confier sa santé à un programme de ce type. Par contre, les techniques de ce type peuvent trouver des applications beaucoup plus utiles dans les programmes éducatifs ou de formation. Plus un programme éducatif répondra de manière souple, variée et intelligente aux réactions de l'utilisateur et plus le succès sur le plan pédagogique sera grand.

Lorsque nous parlons ici de programmes intelligents, cela ne se justifie que par rapport aux 'programmes idiots'. On est encore très loin de l'intelligence au sens où on parle d'intelligence humaine. Mais il faut savoir que des recherches intensives se déroulent actuellement dans le domaine de ce qu'on appelle l'intelligence artificielle.

Le fait de connaître de nombreux faits ainsi que leurs tenants et aboutissants constitue une partie essentielle de l'intelligence. L'accès efficace aux connaissances dont on dispose représente notamment un problème très important. La prochaine leçon essaiera modestement d'aborder ce sujet. Jusqu'ici, les progrès dans le domaine de l'intelligence artificielle restent concentrés dans des domaines relativement étroits. Le langage de programmation le plus adapté au travail dans ce domaine est LISP. Le LOGO est un dérivé de LISP.

Exercices

1) Le nom, le domicile, le sexe, l'âge, la profession, le revenu forment une phrase. Une liste contiendra ces phrases de données pour un groupe de personnes. Formulez

un programme qui permettent de rechercher des personnes d'après plusieurs critères au choix (par exemple toutes les personnes de sexe féminin entre 20 et 50 ans).

2) Un nombre décimal entré par l'utilisateur devra être sorti en clair, c'est-à-dire en toutes lettres. Résolvez d'abord le problème pour des nombres à deux chiffres puis pour des nombres de trois chiffres.

3) Complétez le programme de dialogue par une seconde question du type: comment allez-vous? Après que l'utilisateur ait répondu aux deux questions, c'est-à-dire qu'il ait indiqué son nom et comment il va, le programme devra sortir une réaction qui utilise les deux réponses.

Leçon 16: Structures de données en LOGO

Nouveaux éléments de vocabulaire:
thing, list, listp, throw, TOPLEVEL

Programmes:
FILTRE (filtre de nombres primaires), TABLEAU, TABOUT, TABIN, TABPRINT (fonctions de tableau), ENTREE (programme de tableau), DEVINERBETES (Construction et extension de structures d'arbre), RANGER (tri avec index)

Les nombres, les mots et les caractères sont des données. Dans le cas le plus simple, un nom particulier correspond à chaque donnée. Si on forme une phrase à partir de plusieurs mots, les mots pouvant d'ailleurs parfaitement avoir, en LOGO, la signification de nombres, le nom de la phrase correspondra alors à un ensemble de données. Cet ensemble doit présenter une structure déterminée car les données doivent être stockées dans l'ordinateur de façon à ce qu'on puisse y accéder de façon sélective.

Une phrase composée de mots est un exemple simple de liste. Les listes constituent une base très complexe pour les structures de données en LOGO. C'est pourquoi le vocabulaire de base du LOGO ne prévoit pas d'autres structures de données. Les listes sont un instrument très polyvalent qui permet de construire des structures de données très complexes.

Alors que la plupart des versions du LOGO travaillent avec des listes, DR LOGO ainsi qu'une série d'autres versions disposent avec ce qu'on appelle les listes de propriété d'un autre type de listes qui se prête également aux structures de données.

Les mots d'une phrase peuvent être de longueurs très différentes de

sorte qu'il n'est pas possible de prévoir a priori la représentation interne de ces données dans l'ordinateur. Par contre, lorsqu'on travaille avec des nombres, il est possible de fixer dès le départ leur longueur et de ce fait également leur représentation interne. Les données qui occupent dans l'ordinateur une place uniforme et dont le nombre est connu dès le départ peuvent être gérées plus facilement et plus rapidement que les listes. Pour les applications dans lesquelles il s'agit d'effectuer des calculs intensifs avec beaucoup de nombres, les programmes LOGO sont assez lents et sont de ce fait peu employés.

16.1 Listes simples: Filtre de nombres primaires

La forme de liste la plus simple est une phrase composée de différents mots, c'est-à-dire une série d'éléments séparés entre eux par des espaces.

Un programme de calcul de nombres primaires nous en donnera un exemple supplémentaire. Pour cela, on éliminera au fur et à mesure les nombres entiers divisibles, jusqu'à une certaine grandeur, de façon à réaliser ce qu'on appelle un filtre de nombres primaires.

Dans la version classique d'Eratosthenes, on supprime chaque fois les multiples d'un nombre primaire qu'on vient de trouver. En commençant par 2, ce seront donc tous les nombres pairs, puis les multiples de 3, 5, etc... Si on cherche les nombres premiers jusqu'à 100, le travail est déjà terminé une fois qu'ont été éliminés les multiples de 7. En effet, un nombre qui n'est pas un nombre primaire doit être un multiple d'au moins deux facteurs primaires, le plus petit devant nécessairement être en dessous de 10. C'est pourquoi en règle générale la suppression des multiples n'aura besoin d'être effectuée que jusqu'à la racine carrée du nombre le plus grand.

Nous allons maintenant composer une liste des nombres supprimés. Pour déterminer si un nombre entier est un nombre primaire, il suffira alors de consulter la liste des nombres éliminés. Hormis 2, tous les nombres primaires sont impairs et on peut donc tout de suite exclure les nombres pairs.

```
to FILTRE :N;NOMBRES PRIMAIRES JUSQU'A N
(type 2 ",)
make "PRIM []; LISTE DES NOMBRES ELIMINES
make "I 1
make "W SQRT :N
repeat (:N/2-1) [make "I :I+2 !
if not memberp :I :PRIM [(type :I ",)!
if :I < :W [ELIMINER :I :I+:I]]]
end

to ELIMINER :J :Z
if :J > :N [stop]
make "PRIM se :PRIM :J
ELIMINER :J+Z :Z
end
```

(La fonction SQRT de calcul de la racine carrée sera présentée plus bas.)

La procédure ELIMINER élimine tous les multiples impairs de J, c'est-à-dire qu'elle les ajoute à la liste PRIM en tant que mots. Les multiples sont produits par addition lors de l'appel récursif. Dans cette version, les nombres sont éliminés plusieurs fois, 15 par exemple en tant que multiple de 3 puis de 5. L'appel avec FILTRE 500 produira par exemple 240 entrées alors qu'il n'y a en réalité que 163 entrées différentes. Un test devrait donc nous permettre d'interdire qu'un même nombre soit introduit plusieurs fois dans la liste. La troisième ligne de ELIMINER devra donc se présenter ainsi:

```
if not memberp :J :PRIM [make "PRIM se :PRIM :J]
```

La liste sera ainsi plus petite mais le temps d'exécution augmentera légèrement à cause du test supplémentaire avec memberp. Sachez par ailleurs que toute l'opération pourrait encore être améliorée mais nous ne le ferons pas car cela rendrait le programme nettement plus compliqué.

La détermination des nombres primaires se fait dans le programme FILTRE en utilisant une solution itérative. Pour qu'on n'examine

que les nombres impairs, la grandeur I est chaque fois augmentée de deux. Si le nombre examiné a déjà été éliminé, le mot de test memberp fournira la réponse oui (TRUE). not transformera cette réponse en FALSE et le reste de la ligne repeat ne sera pas exécuté. Si le nombre n'a pas été éliminé, not memberp fournira la réponse TRUE. Dans ce cas, on aura trouvé un nombre premier qui sera alors sorti. On enclenchera alors l'élimination des multiples avec la procédure ELIMINER. La seconde entrée de ELIMINER détermine le pas de progression dans l'élimination. On entre le double du nombre premier pour que les nombres pairs soient ignorés.

ELIMINER n'élimine pas à vrai dire les nombres mais il les enregistre au contraire. On pourrait bien sûr procéder inversement et former au début une liste de tous les nombres possibles, par exemple les nombres impairs. On noterait ensuite dans cette liste tous les nombres qui ne sont pas des nombres premiers. On pourrait par exemple prévoir au départ un caractère attribué à tous les nombres impairs et qui serait modifié lorsqu'il s'agirait des les éliminer. On ne testerait pas si un nombre a été déjà éliminé avec memberp mais en examinant chaque fois le caractère attribué à un nombre donné. Nous vous laissons le soin de réaliser une version de ce type à titre d'exercice.

DR LOGO sur les ordinateurs AMSTRAD est malheureusement très pauvre en fonctions mathématiques intégrées puisque même une opération telle que la racine carrée, qu'on trouve pourtant sur toutes les calculatrices de poche, n'est pas disponible.

Les racines carrées sont calculées d'après la procédure itérative classique d'après Heron-Newton.

```
to SQRT :x;calcul de la racine carrée
op MOYEN (1+:x)/2
end

to MOYEN :w
make "v (:w+:x/:w) / 2
if :w-:v < 1 [op :v] [op MOYEN :v]
end
```

Ce programme vous a déjà été présenté à la leçon 12. La récursion est ici interrompue dès que des approximations successives se différencient de moins de un car, en ce qui concerne le filtre de nombres premiers, seuls comptent les chiffres avant la virgule.

16.2 Les tableaux: Logoplan

Au lieu de noter dans une liste tous les multiples de nombres premiers, on pourrait par exemple produire un ticket de caisse. On pourrait par exemple calculer le total et des sous-totaux à partir des prix énumérés.

Il serait encore plus intéressant de placer plusieurs tickets de caisse à côté les uns des autres, pour former par exemple la liste des dépenses d'un budget mensuel. On peut ainsi établir des relations croisées entre les différentes données qui peuvent se révéler très utiles pour une planification rationnelle des dépenses.

Il faut pour cela que les dépenses de chaque mois soient chaque fois entrées dans les listes dans un ordre déterminé à l'avance. Lorsqu'aucune dépense n'a été effectuée pour un poste déterminé, on entrera alors une marque appropriée, en général le nombre 0. Avec tous ces éléments, on obtiendra ainsi un tableau en deux dimensions dans lequel les lignes représenteront des postes déterminés, par exemple l'alimentation ou les coût du véhicule alors que les colonnes correspondront aux différents mois.

On placera dans la première colonne d'un tel tableau un terme explicatif pour chacune des lignes de façon à ce que leur signification apparaisse bien. On disposera de même des titres dans la première ligne. Une autre colonne sera utilisée pour présenter la somme annuelle de toutes les dépenses d'un poste donné et la dernière ligne indiquera la somme mensuelle de toutes les dépenses. On pourrait aussi imaginer de prévoir des colonnes ou des lignes pour des indications de pourcentage.

Dans une telle application, l'ordinateur a d'abord une fonction de gestion. Il permet d'imprimer, de modifier, de sauvegarder et donc

de conserver les tableaux. L'ordinateur peut en outre être utilisé pour effectuer des calculs comme par exemple pour l'addition des dépenses mensuelles ou annuelles. On peut ainsi faire rectifier sans problème toutes les valeurs calculées chaque fois que l'on modifie une valeur du tableau.

Les programmes conçus pour de telles applications sont qualifiés de programmes de tableaux de calcul. Les systèmes de programmes professionnels Visicalc ou Multiplan en sont des exemples connus. Ces programmes ont d'ailleurs des fonctions beaucoup plus complexes que celles qu'il nous est possible de décrire ici.

Comment représenter des tableaux sous LOGO?

La liste des dépenses mensuelles est une phrase, c'est-à-dire à nouveau la forme simple d'une liste. Le tableau est aussi une liste mais ses éléments ne sont pas des mots mais des phrases. Il s'agit donc d'une liste composée de listes. Il peut également arriver, avec les tableaux, qu'un élément du tableau contienne déjà plusieurs mots. Nous n'étudierons cependant ici, pour commencer, que le cas où une entrée peut être représentée par un mot, ce qui est aussi le cas des nombres.

Fonctions de tableau

Pour travailler avec des tableaux, on a besoin de procédures qui exécutent des opérations que l'on rencontrera fréquemment.

Un tableau doit être constitué comme une liste spéciale et chaque colonne doit comporter un même nombre de lignes. Tous les points du tableau doivent exister pour qu'on puisse accéder correctement aux données. Là où il n'y a aucune entrée, il faut que figure un caractere de remplacement. La procédure TABLEAU remplit ces conditions.

```
to TABLEAU :NOM :Z :S;CREATION D'UN TABLEAU
make :NOM []
repeat :S [make :NOM fput PLACEVIDE :Z thing :NOM]
end
```

```
to PLACEVIDE :Z
local "COL make "COL []
repeat :Z [make "COL se ". :COL]
op :COL
end
```

La procédure PLACEVIDE fournit en résultat une colonne vide. Il s'agit à nouveau d'une liste composée de Z points, le point remplaçant ici l'entrée manquante.
Essayez:

```
PLACEVIDE 10
```

10 colonnes vides de ce type seront réunies par TABLEAU dans une liste dont le nom sera transmis par NOM comme valeur.

Notez que le nom du tableau est transmis comme valeur d'une grandeur. NOM contient le mot qui doit désigner le tableau. Avec make, le nom ne doit pas être précédé de guillemets mais d'une double point. Comparez:

```
make "NOM ". make :NOM ".
```

Avec les guillemets, un nom appelé NOM est affecté au point. Avec le double point, on va chercher le mot qui a été rangé sous le nom de NOM, dans notre exemple la valeur d'entrée de TABLEAU. C'est ce mot qui donne finalement le nom effectif.

La valeur du nom ainsi produit ne peut être obtenue en plaçant un autre double point devant. Au lieu de cela, c'est le mot LOGO thing qu'il faut utiliser.

```
thing :NOM au lieu de ::NOM
```

Les opérations de base pour le traitement des tableaux sont le remplacement ainsi que la lecture d'une valeur dans un emplacement déterminé du tableau.

La lecture d'une valeur est facile à réaliser puisqu'on peut utiliser le mot LOGO item deux fois de suite.

```
to TABOUT :TAB :I :K;LIRE POSITION TABLE
op item :K item :I :TAB
end

to TABIN :TAB :I :K :W;FIXER POSITION TABLE
if :k = 1 [op fput COLIN first :tab :I :W bf :TAB]
op fput first :TAB TABIN bf :TAB :I :K - 1 :W
end

to COLIN :COLONNE :I :W
if :I = 1 [op fput :W bf :COLONNE]
op fput first :COLONNE COLIN bf :COLONNE :I-1 :w
end
```

Il est un peu plus difficile d'entrer une valeur dans le tableau car la version du LOGO dont nous disposons ne possède pas d'opération symétrique à item dans son vocabulaire de base. C'est à cet effet que nous utiliserons la procédure TABIN. Nous y utilisons la procédure auxiliaire COLIN qui sert à entrer une valeur dans une colonne de tableau isolée. Vous remarquerez immédiatement que TABIN et COLIN sont de structure identique car la seule différence est constituée par l'entrée supplémentaire du numéro de colonne dans TABIN.

Il suffit donc d'examiner de plus près le mode de fonctionnement de COLIN. Essayez:

```
COLIN [1 2 3 4 5] 3 "a
```

Résultat: [1 2 a 4 5]

S'il s'agit de remplacer le premier élément, le premier élément de liste présent sera retranché avec bf :COLONNE, dans le seconde ligne de COLIN et la valeur souhaitée sera ensuite placée au début de la liste avec fput. S'il s'agit par contre de remplacer le troisième élément, la troisième ligne de COLIN retranchera le premier élément avec first et transmettra le reste avec bf :COLONNE dans l'appel récursif comme colonne à traiter maintenant.

Une fois que l'entrée voulue aura été placée au début de la colonne

restante ainsi obtenue, la récursion pourra être interrompue. Lors du retour au niveau de départ, les éléments retranchés auparavant seront reinsérés dans leur position d'origine avec fput.

COLIN peut par ailleurs également être employé si les éléments ne sont pas des mots mais des listes:

```
COLIN [a b c d] 2 [5 [a d]]
```

Résultat: [a [5 [a d]] c d]

La procédure TABIN fonctionne de façon similaire mais les éléments sont maintenant constitués par des colonnes entières.

Une autre fonction de base servira à la sortie d'un tableau.

```
to TABPRINT :TAB :I :K :TPOS :TITRE;SORTIR TABLEAU
ct pr :TITRE PU
local "S make "S 1
repeat :K [COLPRINT item :S :TAB :I item :S :TPOS 1 !
make "S :S+1]
end

to COLPRINT :COL :I :H :V
local "Z make "Z 1
repeat :I [setcursor se :H :V + :Z pr item :Z :SP !
make "Z :Z+1]
end
```

TABPRINT sort le tableau TAB, I et K représentent les nombres de lignes et de colonnes. Il est possible de ne sortir que des parties du tableau. On attend sous le nom de TPOS une liste dans laquelle soit fixée la largeur d'impression des différentes colonnes. Un titre est encore prévu pour constituer le titre de l'en-tête du tableau.

TBPRINT utilise la procédure COLPRINT qui ne sort qu'une seule colonne. Les procédures de sortie sont, pour changer un peu, de formulation itérative, c'est-à-dire sans récursion. Les répétitions simples peuvent en effet fort bien être programmées sans récursion.

Pour en finir avec les fonctions de tableau élémentaires, voici maintenant un programme d'entrée qui permet d'entrer et de modifier le tableau TAB.

```
to ENTREE :TAB :TPOS :TITRE;ENTRER TABLEAU TAB
(local "smax "lmax)
make "smax count thing :tab
make "lmax count first thing :TAB;DIMENSION DE TAB
TABPRINT thing :TAB :lmax :smax :TPOS :TITRE
label "DEBUT make "SN 1
catch "HOME [repeat :smax [TBENTREE :SN make "SN :SN+1] !
UNELIGNE 20]
go "DEBUT
end

to TBENTREE :SN
COLENTREE 1 item :SN :TPOS
end

to COLENTREE :L :tp
if :L > :lmax [stop]
(local "pp "e "so "ancien)
make "pp se :tp :l+1
REVERSE setcursor :pp
make "ancien item :L item :SN thing :TAB type :ancien
UNELIGNE 20 make "e rq
if emptyp :e [go "SUITE] if :e=char 94 [PRINT throw "HOME]
if not :e = char 92 [make "ancien :e make :TAB TABIN !
thing :TAB :L :SN :e]
label "SUITE
if :e = char 92 [stop] [COLENTREE :L+1 :tp]
end
```

Avant que le tableau soit prêt pour être modifié, il est imprimé. Commence alors l'entrée colonne par colonne dans la ligne de ENTREE qui commence par

```
catch "HOME
```

Le catch "HOME permet d'interrompre prématurément l'entrée du

tableau sans devoir pour cela repasser dans l'autre sens par tous les niveaux d'appels de procédures.

L'entrée proprement dite se fait dans le programme COLENTREE. Le champ actuellement examiné est d'abord conservé dans la variable "ancien puis sorti à nouveau sur l'écran. Le programme REVERSE, qui ne comporte qu'une instruction est appelé auparavant.

```
to REVERSE
type char 24;CPC
;type word char 27 "p;PCW
end
```

La sortie du caractère ASCII 24 sur le CPC AMSTRAD a pour effet d'échanger pour les sorties suivantes sur l'écran les couleurs de fond et d'écriture, ce qui fait donc ressortir le champ actuel.

Sur le PCW AMSTRAD, cette inversion ne peut être obtenue avec char 24. La représentation inversée sur l'écran y est en effet obtenue avec la séquence Escape ESC p et elle est déconnectée avec ESC q, le symbole ESC étant chaque fois produit à l'aide de char 27.

Lorsque cette zone du tableau est abandonnée, il faut que ce champ retrouve sur l'écran son apparence normale. C'est ce que nous obtenons avec la petite procédure PRINT.

```
to PRINT
type char 24;CPC
;type word char 27 "q;PCW
setcursor :pp type :ancien
end
```

L'entrée effective se déroule dans une zone d'entrée séparée qui est vidée et commandée par la procédure UNELIGNE.

```
to UNELIGNE :i
setcursor se 0 :i
repeat 11 [type char 32]
setcursor se 0 :i
end
```

Outre l'entrée d'une valeur de tableau, trois cas particuliers sont encore prévus:

- Une entrée vide, c'est-à-dire le fait d'appuyer sur la touche RETURN sans entrée, fait passer au champ suivant.
- L'entrée du caractère flèche vers le haut entraîne un saut au début du tableau.
- Le caractère Back Slash (char 92) entraîne le passage à la colonne suivante.

On pourrait naturellement tout aussi bien utiliser d'autres caractères spéciaux, notamment sur le PCW où char 92 ne peut être produit que par la combinaison ALT+q.

Exemple:

```
make "TITRE "BUDGET
make "TPOS [0 10 15 20 25 30 35]

TABLEAU "BUDG1.HJ 4 7
ENTREE :BUDG1.HJ :TPOS :TITRE
```

Vous pouvez maintenant remplir les emplacements qui apparaissent au départ seulement sous forme de points.

```
        JAN  FEV  MAR  AVR  MAI  JUN

ALIMENT.380  410  .    .    .    .
LIVRES  40   62   .    .    .    .
AUTO    210  180  .    .    .    .
```

Notre tableau a ainsi été doté de titres et en partie rempli de nombres. Si on appelle ensuite à nouveau ENTREE, le tableau apparaîtra sous la forme préparée jusqu'ici. On pourra alors continuer à le remplir mais aussi le modifier. Si le programme est sauvegardé sur disquette, le tableau et les indications nécessaires TPOS et TITRE sont sauvegardées avec le programme.

Lors de l'appel de ENTREE, ce n'est pas la valeur mais le nom du tableau qui est transmis. C'est pourquoi il faut accéder au tableau, dans les programmes, avec thing :TAB. La raison tient à l'entrée du tableau avec

```
make :TAB TABIN ...
```

Si on écrivait ici "TAB au lieu de :TAB, le tableau aurait toujours le nom fixe TAB. La solution choisie ici permet au contraire de traiter différents tableaux.

Le programme sera encore amélioré à la leçon suivante de façon à ce que des calculs à l'intérieur du tableau puissent également être prévus.

16.3 Structures d'arbre: Devinette de noms d'animaux

Supposons que vous jouiez aux cartes et que votre partenaire vous demande de choisir une carte. Comme il a peu de chances de deviner la carte choisie par simple supposition, il vous posera des questions orientées.

Il serait bien sûr enfantin de disposer les cartes dans l'ordre puis de vous demander pour chacune si c'était celle-ci. L'art de votre partenaire doit consister au contraire à poser le moins de questions possibles tout en parvenant cependant au but recherché. Il essaiera souvent de dissimuler la méthode utilisée. Dans un jeu loyal, la première question pourrait être: la carte est-elle rouge? Après la couleur, il pourrait encore vous demander le chiffre ou la figure.

Dans un autre jeu, il s'agira de deviner une langue. Le dialogue pourrait se dérouler ainsi:

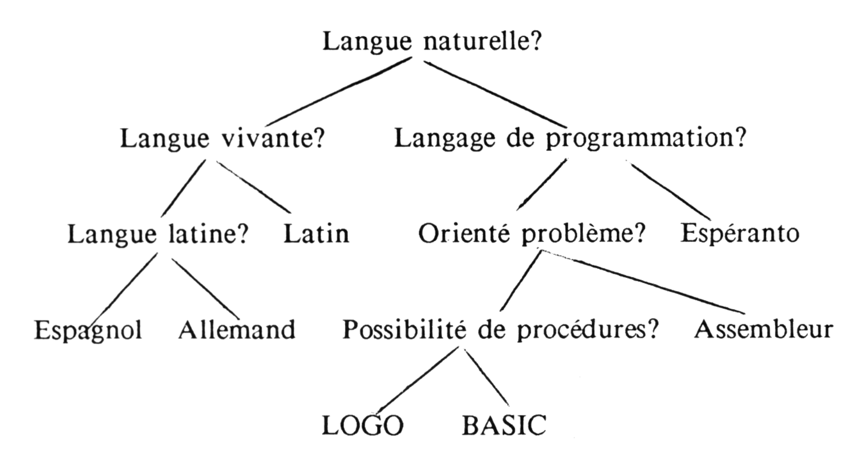
Cette langue est-elle une langue naturelle? - Non. Est-ce un langage de programmation? - Non. Est-ce l'espéranto? - Oui.

Si on acquiesce à la question sur le langage de programmation, le dialogue pourrait se poursuivre ainsi:

Ce langage est-il orienté sur les problèmes? - Oui.
Les procédures sont-elles possibles? - Oui.
S'agit-il du LOGO? - Oui.

Dans ces dialogues, celui qui pose les questions parcourt un arbre de décision. Si, comme ici, seules des réponses Oui/Non sont possibles, on utilise ce qu'on appelle un arbre binaire. On n'utilise pas de tels arbres uniquement dans des situations de jeu. En effet les arbres de décision fournissent des méthodes efficaces de sauvegarde et de recherche de données pour les programmes de banques de données.

La structure de recherche d'une langue peut être représentée graphiquement de la façon suivante:



Dans un arbre il y a ce qu'on appelle des noeuds à partir desquels des branchements peuvent être effectués. Dans un arbre du type binaire que nous venons d'examiner, un noeud débouche précisément sur deux possibilités, l'une pour la réponse oui, l'autre pour la réponse non. Chacune de ces deux possibilités peut à son tour être un noeud avec branchements, par exemple une question, ou bien un noeud final, par exemple le nom d'une langue.

Cette structure de données peut être décrite immédiatement avec des listes:

```
[[langue naturelle] [[langue vivante] [[langue latine]
espagnol allemand] latin] [[langage de programmation]
[[oriente probleme] [[procedures possibles] LOGO BASIC]
Assembleur] Esperanto]
```

La question (noeud avec branchement) et ses deux branches (branche oui et branche non) donnent chaque fois une liste de trois éléments. Vous pouvez le vérifier dans la liste ci-dessus à l'aide de COUNT! Comme la question peut être une phrase de plusieurs mots, elle est, en tant qu'élément, elle-même une liste simple. Si les branches elles-mêmes sont des noeuds avec branchements, ces éléments de liste se composent à leur tour de listes. On peut de même représenter également des arbres qui aient plus de deux branches.

La notion de liste ouvre la possibilité de construire une hiérarchie d'éléments imbriqués les uns dans les autres, ce qui correspond exactement à une structure d'arbre. Les tableaux examinés précédemment peuvent également être interprétés comme des structures d'arbre. Le premier noeud pourrait être une des colonnes existantes et on pourrait, dans le second niveau, sélectionner chaque fois une ligne. La structure d'arbre peut cependant être utilisée de façon plus diversifiée car certaines sections se terminent plus tôt que d'autres qui peuvent poursuivre leurs branchements.

La structure d'arbre est une organisation efficace pour pouvoir rechercher une information voulue. Un arbre peut être modifié et prolongé. C'est pourquoi la structure d'arbre convient particulièrement là où l'on a affaire à une masse d'informations pouvant être complétées ou modifiées de façon dynamique. Le programme d'exemple suivant nous permettra d'illustrer le maniement de cette structure de données.

Devinette de noms d'animaux

La devinette de noms d'animaux est un programme favori de démonstration de l'organisation des structures d'arbre. On utilise volontiers les noms d'animaux comme exemple de données du fait que

le monde animal est si considérablement diversifié et forme par là-même des arbres à nombreuses ramifications.

Si le programme travaillait avec un arbre de décision fixé une fois pour toutes, il deviendrait vite ennuyeux. Les programmes qui permettent d'accroître l'arbre de connaissance grâce aux réponses de l'utilisateur sont beaucoup plus attrayants.

La forme d'arbre la plus simple est un noeud final unique, en l'occurrence donc un nom d'animal:

```
"singe
```

Le programme ne peut alors poser que la question:

```
Pensez-vous au : singe?
```

Si la réponse est non, le programme ne s'en satisfera pas et il demandera qu'on lui indique le terme qu'il n'a pas su deviner. Pour être mieux armé pour le prochain tour, on demandera à l'utilisateur d'indiquer une question qui permette de distinguer le terme auquel il pensait du terme singe proposé. Le plus simple est alors d'indiquer si le nouveau terme doit correspondre à la branche Oui ou à la branche Non. La liste présentera ensuite par exemple la structure suivante:

```
[[peut-il voler?] colibri singe]
```

Le programme doit donc passer du stade:

de singe (un mot)

à celui de [[peut-il voler?] colibri singe] (une liste).

Dans les deux cas, le mot SINGE est le dernier élément.

On peut déjà en déduire la méthode à utiliser pour compléter les connaissances. Les connaissances doivent être complétées lorsqu'un noeud final a été atteint et rejeté par la réponse non. Ce noeud final, ici le mot singe, doit être remplacé par une liste de trois

éléments, la question, le terme nouveau et le terme ancien.

La nouvelle liste a maintenant deux noeuds finaux, colibri et singe. Dans la prochaine étape de l'extension, l'un des deux devra être remplacé par une nouvelle liste. Pour que cela reste possible même avec une liste d'une structure compliquée, il faut d'abord rechercher le terme à remplacer. Fondamentalement, ce problème a déjà été examiné pour le remplacement de mots dans des textes.

Constitution de listes

Avant que les procédures d'extension d'un arbre de connaissance puissent être examinées, il nous faut compléter les possibilités d'organisation de listes sous LOGO.

Les listes se reconnaissent aux crochets qui les enferment. On peut former des listes à partir de mots sous la forme simple de phrases. Pour compléter une phrase par d'autres mots, on emploie le mot LOGO se:

```
se [Une [phrase complete]
```

Réponse: `[Une phrase complete]`

```
se "Une [phrase complete]
```

Réponse: `[Une phrase complete]`

On entre dans le second exemple un mot et dans le premier une liste composée d'un seul mot. se traite les mots comme des listes et réunit toutes les entrées en une liste. L'effet de se peut donc être ainsi décrit:

> Les crochets autour des entrées sont d'abord supprimés s'il y en a. Tout est ensuite réuni et le résultat est à nouveau placé entre crochets.

Cependant, seuls les crochets extérieurs sont supprimés. Pour les listes imbriquées, les crochets de l'intérieur subsistent.

```
se [Une] [phrase [complete]]
```

Réponse: [Une phrase [complete]]

Vous connaissez la seconde possibilité d'organisation des listes que constituent les mots fput et lput. fput aura en général un effet différent de se.

> Les crochets de la seconde entrée sont d'abord supprimés. La première entrée est ensuite placée devant, sans modification. Le résultat est ensuite à nouveau placé entre crochets.

```
fput "Une [phrase complete]
```

Réponse: [Une phrase complete]

```
fput [Une] [phrase complete]
```

Réponse: [[Une] phrase complete]

Avec fput, ce n'est pas la même chose que la première entrée soit un mot ou une liste composée d'un seul mot. lput fonctionne exactement comme fput si ce n'est que l'entrée est ajoutée à la fin.

A côté de se et fput (lput), il existe encore une autre possibilité d'organiser les listes, le mot LOGO list.

```
list "Une "phrase
```

Réponse: [Une phrase]

```
list "Une [phrase]
```

Réponse: [Une [phrase]]

```
list [Une] [phrase]
```

Réponse: [[Une] [phrase]]

```
list [Une] [phrase [complete]]
```

Réponse: [[Une] [phrase [complete]]]

L'effet de list est donc le suivant:

> Les entrées sont placées sans modification à la suite les unes des autres puis le résultat est placé entre crochets.

Le mot de test wordp permet de déterminer si on a affaire à un mot. Inversement, il existe aussi un mot de test listp. listp répondra TRUE si l'entrée est une liste, sinon FALSE.

```
listp "MOT
```

Réponse: FALSE

```
listp 5
```

Réponse: FALSE

```
listp [LISTE]
```

Réponse: TRUE

Procédure d'extension de l'arbre de connaissance

La tâche principale consiste à remplacer un noeud final. Le noeud sera alors un mot alors que la liste de remplacement se composera d'une phrase d'interrogation, d'un nouveau terme et de l'ancien mot. On peut employer pour cela le mot LOGO list.

```
(list [peut-il voler?] "colibri "singe )
```

On suppose ici que le mot singe existe déjà et qu'il est par exemple disponible sous le nom noeud. La question peut être lue comme une liste avec rl et le nouveau terme avec rq. Essayez:

```
make "noeud "singe (list rl rq :noeud)
```

Pour l'entrée au clavier, vous devez d'abord entrer la question, la terminer avec RETURN puis tapez ensuite le nouveau terme y compris RETURN.

Au niveau suivant, cela devient plus difficile car c'est maintenant un des deux noeuds finaux, colibri ou singe, qui doit être remplacé par une liste de trois éléments. Pour cela, le second élément doit être remplacé dans la liste existante si nous nous trouvons dans la branche Oui, c'est-à-dire si la question a reçu précédemment une réponse positive. Dans la branche Non, c'est par contre le troisième élément qui devra être remplacé.

```
make "arbre !
(list first :arbre (list rl rq item 2 :arbre) last :arbre)
```

Si vous entrez maintenant:

```
Est-il grand (RETURN) aigle (RETURN)
```

lors du traitement de la seconde ligne, arbre devrait ensuite avoir la valeur suivante:

```
[[peut-il voler] [[est-il grand] aigle colibri] singe]
```

Lorsqu'il s'agit de remplacer le troisième élément dans la branche Non, on peut utiliser la ligne:

```
(list first :arbre item 2 :arbre (list rl rq item 2 :arbre))
```

ou un peu plus court:

```
lput (list rl rq last :arbre) piece 1 2 :arbre
```

Le dialogue pour trouver le terme à deviner et éventuellement pour compléter l'arbre de connaissance devrait maintenant être effectué en gros de la façon suivante:

- Sortie du premier élément de la liste, c'est-à-dire de la première phrase (Peut-il voler?).

- Suivant la réponse (Oui ou Non), on prendra le second ou le troisième élément comme noeud final et on demandera si le terme trouvé est le bon.

- En cas de réponse négative, le terme rejeté devra être remplacé par la liste que nous avons décrite et qui se compose d'une nouvelle question, d'un nouveau terme ainsi que de l'ancien terme.

- En cas de réponse positive, le terme recherché est présenté comme résultat.

```
to COMPLETER :arbre
(type first :arbre "? )
if lc first rq = "o
[op (list first :arbre QUESTIONS item 2 :arbre last !
:arbre)]
[op lput QUESTIONS last :arbre piece 1 2 :arbre]
end

to QUESTIONS :noeud
(type [pensez-vous a: ] :noeud "?)
if lc first rq="o [pr [alors j'avais trouve ]!
op :noeud]
type [je me suis trompe ! A quoi pensiez-vous?]
make "nouveau rq
(pr [Entrez une question appelant la reponse oui pour]!
:nouveau [et non pour] :noeud [ !]
op (list rl :nouveau :noeud)
end
```

Là où dans la première version nous avions écrit (list rl rq last :arbre), nous plaçons maintenant le résultat de la procédure QUESTIONS. Il ne faut pas en effet que l'utilisateur tape les questions à l'aveuglette. Si le terme a effectivement déjà été trouvé, QUESTIONS fournira simplement en réponse le noeud entré.

Dans ce cas, COMPLETER ne complètera pas en fait l'arbre de

connaissance car dans la forme proposée, le noeud sera remplacé par lui-même. On pourrait faire ici l'économie de ce remplacement en effectuant un test à cet endroit. Cependant la formulation proposée est plus facile à généraliser.

La procédure COMPLETER ne fonctionne jusqu'ici que lorsque arbre se compose exactement d'une question et de deux noeuds finaux. Aux niveaux supérieurs, il faudra poser plusieurs questions pour que l'arbre soit parcouru jusqu'à une question unique et à deux noeuds finaux.

Comme vous le supposez déjà certainement, la généralisation sera obtenue grâce à une récursion. Dans COMPLETER, QUESTIONS sera remplacé par l'auto-appel, c'est-à-dire par COMPLETER. La récursion devra être interrompue lorsque COMPLETER rencontrera un noeud final. Cela peut aisément être déterminé avec le mot de test wordp. Lorsqu'on a effectivement rencontré un noeud, celui-ci est transmis à QUESTIONS. Comme au bout du compte le résultat de QUESTIONS doit être placé dans l'arbre, le résultat de QUESTIONS sera maintenant renvoyé comme résultat de COMPLETER au plus bas niveau d'imbrication.

```
to COMPLETER :arbre
if wordp :arbre [op QUESTIONS :arbre]
(type first :arbre "? )
if lc first rq = "o
[op (list first :arbre !
COMPLETER item 2 :arbre last :arbre)]!
[op lput COMPLETER last :arbre piece 1 2 :arbre]
end

to DEVINERBETES
(pr [Pensez a un nom de] "bete!
[Je vais essayer de le deviner ])
make "connaissances COMPLETER :connaissances
DEVINERBETES
end
```

DEVINERBETES est le programme-cadre. Dans la première ligne de salutation, nous avons volontairement placé bete en dehors des

crochets. Vous pourriez en effet faire deviner d'autres types de termes. Avec POCALL "DEVINERBETES, le déroulement du programme apparaît nettement:

```
0 : DEVINERBETES
1 : ...COMPLETER
2 : ......QUESTIONS
2 : ......COMPLETER*
2 : ......COMPLETER*
1 : ...DEVINERBETES*
```

Lors de chaque parcours, l'arbre de connaissance est à nouveau affecté au nom connaissances. L'arbre peut donc être sorti et sauvegardé comme une structure de liste. connaissances est un nom global, on suppose ici qu'il existe déjà au moins un noeud. Pour deviner des bêtes, on peut par exemple lancer le programme ainsi:

```
make "connaissances "singe DEVINERBETES
```

On peut toutefois aussi commencer avec un arbre déjà développé sous le nom de connaissances, par exemple avec l'arbre développé au début de cette section pour les langues.

Lorsqu'on veut construire un arbre avec un terme de départ existant déjà dans une première version, on doit procéder ainsi: il faut chaque fois entrer à nouveau le terme qui devra apparaître au prochain niveau dans la branche Non. Le programme demande en effet toujours une nouvelle question appelant une réponse positive pour le nouveau terme.

16.4 Données liées: tri avec clés

Dans un fichier de cartes, on écrit généralement plusieurs données de types différents sur chaque carte du fichier. Dans un fichier du personnel, ce seront par exemple le nom, le domicile, le code postal, la rue, le sexe, la date de naissance et le lieu de naissance, la profession, la catégorie de salaire, etc... Ces données sont pour partie des chaînes de caractères, parfois aussi des nombres inscrits dans une structure donnée, comme par exemple

la date de naissance. Ces données peuvent comprendre un ou plusieurs éléments.

Toutes les données sont inscrites sur une carte du fichier et elles sont solidaires. C'est pourquoi on parle de données liées. Lorsqu'on utilise le fichier, cela affectera parfois la totalité de la carte du fichier, par exemple si on la supprime. Il arrivera parfois également qu'on ne veuille pas traiter la totalité des données liées et qu'on ne veuille accéder qu'à des parties isolées.

La notion de liste est suffisamment souple pour qu'on puisse manier de telles structures de données. Peu importe pour une structure de liste que les données partielles soient de différentes types et qu'elles se composent éventuellement elles-mêmes de plusieurs parties. L'ensemble d'une carte du fichier est représenté comme une liste dont les différentes entrées sont des éléments de la liste et peuvent s'il en est besoin être à leur tour des listes.

On accède à l'ensemble de la carte du fichier à l'aide du nom de liste. Pour cela, la liste doit être décomposée de la manière appropriée puis à nouveau recomposée. Plusieurs solutions sont possibles pour cela. Nous allons vous présenter une variante possible qui sera illustrée par un exemple de tri avec une clé de tri.

Lorsqu'il s'agit de trier un fichier, cela peut être effectué à différents points de vue. Pour un fichier de personnes, on souhaite le plus souvent un tri alphabétique d'après le nom. Mais un tri d'après la date de naissance ou d'autres caractéristiques personnelles intéressantes est également envisageable. La grandeur qui servira à fixer l'ordre des données est appelée clé de tri. Lors du tri d'une colonne de nombres en ordre croissant, les données sont identiques à la clé. Dans un fichier de personnes, la clé ne représente qu'une partie d'une carte du fichier. Mais évidemment les cartes doivent bien sûr être classées malgré tout chacune dans sa totalité lors du tri.

Pour accéder à la clé, nous formulerons une procédure CLE:

```
to CLE :l
```

```
op first :l
end
```

Ici la clé sera tout simplement le premier élément de l'entrée. Dans d'autres cas, le résultat pourra être isolé de l'entrée avec item par exemple. Si les données et la clé sont identiques, l'entrée sera renvoyée comme résultat sans modification.

Une fois cette procédure formulée, d'autres programmes, ici le programme de tri, peuvent y accéder sous le nom CLE. Si la clé du tri doit être modifiée, seule la procédure CLE en sera affectée. Une structure de données déterminée est représentée avec cette méthode par des procédures de ce type.

Le moyen le plus simple de lier des données sous LOGO est représenté par ce qu'on appelle les listes de propriétés. Les propriétés peuvent alors être utilisées comme clé. Lorsqu'on change la propriété utilisée à cet effet, l'accès aux données est également modifié en conséquence. La procédure CLE devient alors:

```
to CLE :l
op gprop :l :notion
end
```

Sous le nom notion doit être alors disponible la particularité clé (Nom, ville etc...) sous forme de nom global ou bien la notion clé doit être indiquée ici directement avec

```
op gprop :l :nom
```

La construction de telles structures de données avec des listes de propriétés sera traitée dans une leçon spécifique qui contient une gestion de fichier complète.

Le tri

Il existe une littérature abondante sur les méthodes de tri. La méthode optimale dépend de la masse de données à trier ainsi que de son rangement (ou plutôt son degré de "désordre"). Nous utiliserons

ici une méthode simple appelée insertion linéaire. On rencontrera rarement de très grandes masses de données dans des programmes LOGO car les programmes LOGO travaillent encore trop lentement pour cela pour le moment. L'insertion linéaire est donc généralement à préférer aux méthodes plus sophistiquées qui exigent bien sûr également un plus gros travail de programmation.

La procédure suivante, LINSORT, attend comme entrée une liste ou plus exactement une phrase qu'elle sortira ensuite triée dans un ordre croissant.

```
to LINSORT :fichier
op LIN (list first :fichier) bf :fichier
end

to LIN :v :h
if emptyp :h [op :v]
(pr :v "-::h);Sortie pour test
op LIN IN :v first :h bf :h
end

to IN :l :e
if emptyp :l [op (list :e)]
if first :l < first :e !
[op fput first :l IN bf :l :e] [op fput :e :l]
end
```

Pour bien comprendre la méthode utilisée, le mieux est que vous commenciez par l'essayer:

```
LINSORT [42 53 10 40 92 16 4 65]
```

Dans la troisième ligne de LIN, nous avons placé une ligne de test qui sortira chaque fois l'état actuel de la liste à trier. Vous devriez donc voir apparaître les lignes suivantes à l'écran:

```
LINSORT [42 53 10 40 92 16 4 65]
42 - 53 10 40 92 16 4 65
42 53 - 10 40 92 16 4 65
10 42 53 - 40 92 16 4 65
```

```
10 40 42 53 - 92 16 4 65
10 40 42 53 92 - 16 4 65
10 16 40 42 53 92 - 4 65
4 10 16 40 42 53 92 - 65
[4 10 16 40 42 53 65 92]
```

La procédure LIN a deux entrées v et h qui représentent respectivement la partie antérieure ou postérieure de la liste. La partie antérieure v est déjà triée et elle s'accroît jusqu'à contenir toute la liste.

L'insertion linéaire consiste à retrancher chaque fois le premier élément de la partie postérieure, celle qui n'est pas triée pour l'insérer dans l'emplacement qui convient de la partie antérieure déjà triée. Cette opération peut être formulée directement sous forme récursive.

L'insertion de l'élément retranché se fait dans la procédure IN. Cette opération peut également aisément être décrite de façon récursive: l'élément à insérer est comparé au premier élément de la liste. S'il est plus petit, il peut être placé directement au début. Dans le cas contraire on détache les éléments placés au début jusqu'à ce qu'on trouve effectivement un élément plus grand ou jusqu'à ce que la liste soit épuisée. L'élément à insérer sera donc toujours placé au début de la liste restante qui peut fort bien être vide. La liste est ensuite recomposée de la façon habituelle.

LINSORT permet également de classer des mots par ordre alphabétique:

```
LINSORT [Goethe Erasme Bachelet]
```

Résultat: [Bachelet Erasme Goethe]

La procédure IN est déjà formulée de telle façon que la liste à trier soit correctement recomposée même si les éléments de cette liste sont eux-mêmes des listes. Cela n'a bien sûr d'intérêt qui si on utilise une clé de tri. La troisième ligne de IN devra alors commencer par

```
if CLE first :l < CLE first :e
```

Préparons en guise d'exemple un fichier de noms

```
make "fichier [[Nils Bohr] [Albert Einstein] [Otto Hahn]
[Friedrich Hund]]
op first :l in CLE:
[[Albert Einstein] [Friedrich Hund] [Nils Bohr] [Otto
Hahn]]
```

Peut-être voulez-vous connaître la date de naissance de ces grands physiciens? Vous pourrez ensuite faire effectuer un tri d'après cette date de naissance.

```
[[Otto Hahn 1879] [Albert Einstein 1879] !
[Nils Bohr 1885] [Friedrich Hund 1894]]
```

Exercices

1) Ecrivez un programme utilisant FILTRE pour rechercher ce qu'on appelle des nombres primaires jumeaux. Les nombres primaires jumeaux sont des nombres primaires voisins dont la différence n'est pas supérieure à deux.

2) Dans un examen, par exemple pour passer le permis de conduire, on devra résoudre un certain nombre d'exercices. Le traitement de chaque exercice est noté par des points. Utilisez un programme de tableau comme celui de la leçon 16 pour évaluer un examen de ce type pour 10 participants.

3) Une expression de la forme A*(B+C) peut être représentée sous la forme d'une structure d'arbre. La liste correspondante serait:

```
[* [+ B C] A]
```

Représentez l'expression (A+B)*C+D*(A-E). Essayez d'utiliser le programme DEVINERBETES pour construire des structures d'arbre de ce type.

Leçon 17: Les programmes en tant que listes

Nouveaux éléments de vocabulaire:
text, define, run, Structure de liste de programmes

Programmes:
LOGOPLAN (programme de table de calcul avec programmes auxiliaires), GENERATEURDEMASQUE (générateur de programme)

17.1 Texte de programme

Les programmes LOGO sont formulés à l'aide de mots LOGO qui font partie du vocabulaire de base ou qui ont été définis par l'utilisateur lui-même. Les données apparaissant dans les programmes sont également composées de mots. De même, les nombres sont des mots pour le LOGO. Les programmes et les données sont donc formés avec les mêmes composants.

Les programmes ont une structure: le texte du programme est organisé en lignes qui correspondent aux phrases dans un texte. Il va donc pratiquement de soi que le LOGO représente la structure des programmes sous forme de listes.

Le texte du programme peut être sorti avec po et il apparaît alors comme vous l'avez écrit, même s'il peut arriver que des espaces soient encore ajoutés par le système LOGO. Essayez la phrase suivante:

```
po "CARRE text "CARRE
```

Vous pouvez bien sûr prendre n'importe quelle autre procédure à la place de CARRE.

```
to CARRE :cote
repeat 4 [fd :cote rt 90]
end

[[cote] [repeat 4 [fd :cote rt 90]]]
```

La procédure LOGO text a sorti le texte de programme sous la forme d'une liste dont les éléments sont les lignes de programme qui se présentent elles-mêmes comme des listes. Le nom de procédure en tête de la procédure n'apparaît pas car c'est le nom qui a été donné au texte de programme.

Le premier élément de liste est la liste des entrées. Si une procédure n'a pas d'entrées, LOGO place ici une liste vide.

Si on entre avec text un nom de procédure qui n'existe pas, c'est une liste vide qui est renvoyée comme texte de programme.

```
text "PASLA
```

Résultat: []

Le texte du programme peut être doté d'un nom comme n'importe quelle autre liste.

```
make "PROGRAMME text "CARRE
pr :PROGRAMME
```

Le texte de programme de la procédure CARRE sera maintenant sorti comme valeur de PROGRAMME.

La liste produite à l'aide de text peut être traitée comme n'importe quelle autre.

```
first text "CARRE
```

Vous obtiendrez la liste des entrées de la procédure CARRE:

Réponse: [cote]

text permet au départ uniquement d'examiner les procédures mais le texte du programme peut ensuite être manipulé sous la forme d'une liste.

17.2 Les programmes écrivent des programmes

text produit à partir d'une procédure existante une liste avec le texte de programme disposé ligne par ligne. Inversement, on peut faire une procédure à partir d'une liste contenant un texte de programme. Le LOGO nous fournit à cet effet le mot define.

```
make "PROGRAMME [[cote] [repeat 4 [fd :cote rt 90]]]

define "CARRE :PROGRAMME po "CARRE

to CARRE :cote
repeat 4 [fd :cote rt 90]
end
```

La procédure CARRE a effectivement été déclarée à partir du texte de programme approprié, à l'aide de define. Le LOGO ne sort d'ailleurs pas de message 'CARRE defined' comme cela se produit normalement lorsqu'on écrit un programme. Lorsqu'on définit des procédures pendant le déroulement même d'un programme, la sortie écran ne doit pas en effet être perturbée par des messages système.

- define produit une procédure LOGO. Deux entrées sont attendues, la première étant le nom de la procédure à déclarer.

- La seconde entrée doit être une liste organisée ligne par ligne. Chaque élément de cette liste doit être à son tour une liste, c'est-à-dire chaque fois une ligne de programme. Le premier élément se compose de la liste des entrées de procédure. Si aucune entrée n'est prévue, une liste vide doit être mise à la place.

define fonctionne de manière analogue à l'écriture de programmes en mode d'édition avec to ou ed.

Exécution de listes

Chaque liste peut maintenant être inversement interprétée comme texte de programme. De même qu'on peut écrire une ligne de programme et la faire exécuter immédiatement, de même une liste peut-elle également être exécutée immédiatement comme texte de programme à l'aide du mot LOGO run.

```
repeat 5 [pr 1 + random 6]
```

Si cette ligne est écrite comme une ligne d'entrée, elle sera immédiatement exécutée après que vous ayez appuyé sur la touche RETURN. Si elle est définie comme une liste, elle pourra être exécutée comme une ligne d'entrée à travers l'instruction LOGO run.

```
make "ligne [repeat 5 [pr 1 + random 6]]

Exécution: run :ligne
```

run attend en entrée une liste dont le contenu est exécuté comme une ligne d'entrée en mode direct.

run supprime les crochets extérieurs de la liste et en fait une ligne d'entrée qui est exécutée immédiatement. L'entrée pour run peut bien sûr également être le résultat d'une action LOGO.

```
make "action [pr 1 + random 6]
run ( list "repeat 4 :action )
```

Il est ainsi également possible de transmettre des noms de procédure comme entrées d'autres procédures. Ces procédures peuvent alors être utilisées avec run.

```
to EXECUTER :procedure
( pr [La procedure est executee] :procedure )
run :procedure
end

to DES
```

```
pr 1 + random 6
end
```

Appel: EXECUTER [DES]

Si un résultat est produit par la liste d'entrée de run, il faut aussi prendre en compte la sortie. run [...] doit alors être utilisé exactement comme l'appel d'une procédure équivalente.

```
to LOTO
op 1 + random 49
end
```

Appel: pr run [LOTO]

17.3 LOGOPLAN: Programme de table de calcul

Le paquet de programmes de gestion de tableau de la leçon précédente va maintenant prendre un aspect plus professionnel. Les programmes de tables de calcul doivent bien sûr permettre, comme leur nom l'indique, de faire des calculs.

Pour le projet LOGOPLAN, nous allons rendre possible la formation de sommes à l'intérieur de colonnes ou de lignes. On devra pouvoir entrer non seulement le texte du tableau ou les valeurs du tableau mais aussi trois instructions:

- Z 2 5: la somme des valeurs de tableau de la colonne appelée sera calculée, en commançant par le second élément et jusqu'au cinquième élément de la cinquième ligne. Cette somme sera inscrite à la place d'une valeur d'entrée.

- S 3 6: on formera de même la somme de la ligne actuelle pour la zone de colonnes indiquée.

- ! (point d'exclamation): on réutilisera le calcul de somme défini lors d'entrées précédentes.

Le plus important est de réaliser d'abord les deux premières extensions, la troisième ne représente qu'une gestion supplémentaire des formules utilisées.

L'évaluation d'une formule de tableau se fera dans la procédure TAB.FORMULE qui sera appelée par la procédure VALEUR.

```
to TAB.FORMULE :S :formule :I :Imax
;EVALUATION D'UNE FORMULE DE TABLEAU
if :I > :Imax [op :S]
op TAB.FORMULE run :formule :formule :I+1 :Imax
end

to VALEUR :E;DECODAGE D'UNE INSTRUCTION
if first :E ="Z [op TAB.FORMULE 0 !
[:S+item :I item :SN thing :TAB] item 2 :E last :E]
if first :E ="S [op TAB.FORMULE 0 !
[:S+item :L item :I thing :TAB] item 2 :E last :E]
end
```

TAB.FORMULE reste générale. Le résultat est produit de façon récursive sous le nom S. La répétition récursive fonctionne pour I de l'élément entré jusqu'à la valeur maximale Imax, par étapes de 1. La formule entrée est évaluée pour chaque valeur de I.

En guise d'exemple pour TAB.FORMULE, vous pouvez essayer l'appel suivant:

```
TAB.FORMULE 0 [:I+:S] 1 5
```

Résultat: 15

Si vous voulez suivre le déroulement des opérations dans le détail, vous devez activer le mode trace. Le résultat sera la somme des nombres naturels de un à cinq. Les deux dernières entrées définissent les limites de l'addition. Avec

```
TAB.FORMULE 1 [:I*:S] 1 5
```

Résultat: 120

vous ferez calculer le produit des nombres naturels de un à cinq, ce qui est noté par 5!. Comme le montrent ces exemples, la première entrée de TAB.FORMULE représente la valeur de départ pour le calcul récursif.

Dans VALEUR, un appel de TAB.FORMULE est prévu pour les cas d'une instruction commençant par Z ou par S. La différence tient ici uniquement à l'échange entre les lignes et les colonnes. La formule appropriée pour l'appel récursif sera pour Z:

```
:S+item :I item :SN thing :TAB
```

Cela signifie que l'élément du tableau de la ligne I et de la colonne SN sera additionné à la valeur de S. Si le résultat est à nouveau utilisé lors de la récursion comme entrée pour TAB.FORMULE, on aboutit ainsi à la sommation des éléments d'une colonne du tableau.

Comment s'effectue maintenant l'évaluation dans TAB.FORMULE?

La formule ou plus précisément la formulation récursive de la formule est transmise comme valeur d'entrée. Elle fait apparaître le nom I qui est local par rapport à TAB.FORMULE. Sont également utilisés les noms SN ou L (numéro de colonne ou de ligne actuel) qui viennent ici de la procédure d'appel COLIN ou TBIN comme noms globaux par rapport à TAB.FORMULE.

L'évaluation s'effectue alors à l'aide du mot LOGO run.

La procédure VALEUR d'évaluation de la formule est utilisée en liaison avec l'entrée de valeurs du tableau. Dans le paquet de programmes de la section 16.2 cela se faisait dans la procédure COLENTREE qui doit donc être modifiée en conséquence.

```
to COLENTREE :L :tp
if :L > :lmax [stop]
(local "pp "e "so "ancien)
make "pp se :tp :L+1
reverse setcursor :pp
make "ancien item :L item :SN thing :TAB type :ancien
```

```
UNELIGNE 20
setcursor [0 20] make "e rl
if emptyp :e [go "SUITE] if :e = (se char 94) [PRINT throw
HOME]
if :e = [!] [make "e item :l item :l item :SN :TABFORM]
if count :e > 1 [make "TABFORM TABIN :TABFORM :L :SN :e
make "e VALEUR :e] [make "e first :e]
if not :e = char 92 [make "ancien :e make :TAB TABIN thing
:TAB :L :SN :e]
label "SUITE
PRINT
if :e = char 92 [stop] [COLENTREE :L+1 :tp]
end
```

L'entrée devra maintenant se faire avec rl. Si on entre seulement le point d'exclamation, l'entrée sera remplacée par la valeur correspondante du talbeau TABFORM. Si la liste d'entrée contient plus d'un élément, c'est qu'il s'agit d'une formule qui sera placée dans le tableau TABFORM. La formule est ensuite évaluée avec VALEUR. Si la liste d'entrée ne comporte par contre qu'un seul élément, celui-ci sera fourni sous la forme d'un mot par first.

Cette version étendue travaille non seulement avec le tableau véritable mais aussi avec un second tableau TABFORM qui est produit exactement comme TAB et qui comporte donc exactement le même nombre d'éléments. Dans ce tableau sont enregistrées uniquement les instructions éventuellement entrées, pour qu'elles puissent être utilisées plus tard. TABFORM est ici considéré comme un nom global, contrairement au véritable tableau. Ce n'est peut-être pas logique mais cela nous évite de devoir énumérer les modifications nécessaires pour les programmes d'appel.

Exemple:

```
make "TITRE "Budget
make "TPOS [0 10 16 22 28 34]

TABLEAU "Maison 4 6
TABLEAU "TABFORM 4 6
ENTREE "Maison :TPOS :TITRE
```

Le tableau pourra par exemple être rempli de la façon suivante:

	1981	1982	1983	1984	SOMME
ALIMENT.	3200	3450	3600	3820	14070
VETEM.	800	850	900	.	.
TOTAL	4000	4300	4500	.	.

Les sommes de la dernière ligne sont produites en entrant Z 2 3, celles de la dernière colonne en entrant S 2 5.

Lors de la prochaine entrée, des valeurs pourront être modifiées. Pour que les formules soient à nouveau évaluées dans les emplacements réservés pour les totaux, il faudra entrer chaque fois F. Il ne sera donc pas nécessaire de définir les instructions lors de chaque exécution. C'est précisément le but de l'introduction du tableau supplémentaire TABFORM.

17.4 Générateurs de programme

define permet de produire des programmes LOGO. Celui qui utilisera plus tard ces programmes ne saura peut-être jamais comment ils ont été produits. Il est donc en fin de compte sans importance que des programmes soient d'abord conçus sur le papier, sur un bureau, ou qu'ils aient été développés au moyen de programmes conçus à cet effet.

On peut bien sûr se demander maintenant si les programmes peuvent vraiment nous aider à réaliser d'autres programmes. Il faut bien en effet que l'idée qui donne naissance à un programme jaillisse de quelque part. Bien que nous ne puissions pas décrire ici en détail toutes les possibilités offertes par les générateurs de programmes, nous allons examiner maintenant avec un exemple simple les propriétés du LOGO qui se rapportent à ce domaine.

Outre define, le mot LOGO run est également important à cet égard. Une suite d'instructions LOGO peut aussi bien être exécutée directement avec run que convertie ensuite en un programme avec define. Une méthode de production de programmes peut donc se présenter ainsi:

- Une série d'actions sont directement exécutées sur l'ordinateur sous la direction d'un programme approprié et ce en utilisant le clavier.

- On peut ce faisant corriger ou même rejeter différentes étapes.

- Un programme séparé est produit à partir de la séquence des étapes acceptées.

Le programme ainsi créé exécute alors automatiquement toutes les étapes qui ont été déclenchées manuellement lors du travail sur l'ordinateur. Cette méthode permet de faire traiter directement les actions du programme voulues. Comme cette opération se déroule sous la direction d'un programme approprié qui doit être d'un emploi aussi pratique que possible, des programmes peuvent même être réalisés par des utilisateurs qui ne sachent pas du tout programmer en réalité.

La méthode décrite ici suppose que les étapes du programme puissent être testées une par une, qu'on puisse y apporter des modifications et qu'on puisse enregistrer les étapes réussies. Toutes ces conditions sont remplies par le LOGO.

Générateur de masque pour la saisie des données

Nous prendrons comme exemple un programme permettant de produire des programmes pour masques d'entrée. Nous poursuivons donc ici l'étude du sujet abordé avec les programmes de la leçon 10 pour l'entrée de données à l'aide de masques d'entrée.

Les masques d'entrée sont des formulaires produits sur l'écran. L'utilisateur est guidé d'une zone d'entrée à la suivante. A la leçon 10, nous avons réglé ce problème à l'aide des programmes MASQUE et ENTREE. Mais le formulaire était alors défini une fois pour toute dans le programme MASQUE. Pour utiliser un autre formulaire, vous devez dessiner votre formulaire sur le papier et formuler ensuite des programmes MASQUE et ENTREE adaptés.

Nous allons donc développer maintenant un programme qui produira ces procédures automatiquement. L'utilisateur pourra ainsi dessiner le formulaire voulu directement sur l'écran. C'est à partir de là que seront produits les deux procédures MASQUE et ENTREE. Une fois ce travail effectué, les deux procédures définies à partir du programme pourront être sauvegardées et réutilisées ultérieurement.

Cet exemple nous permettra également de montrer comment un utilisateur qui ne sait absolument pas programmer peut pourtant créer des programmes grâce à un programme approprié. De tels générateurs de programmes permettent de faire faire aux ordinateurs un pas important dans le sens d'une utilisation pratique par tous.

L'analyse du paquet de programmes donnera:

```
0: GENERATEURDEMASQUE
1: ...MENTREE
2: ......COMMANDE
2: ......MPOS
3: .........MOT
4: ............CGAUCHE
2: ......MENTREE*
1: ...MLISTE
2: ......MLISTE*
1: ...ELISTE
2: ......ELISTE*
```

MENTREE sert à l'entrée du masque par l'utilisateur. MLISTE produit sous forme de liste le texte de programme pour la procédure MASQUE à créer. ELISTE produit alors le texte de programme pour la procédure ENTREE.

```
to GENERATEURDEMASQUE
make "Pliste []
make "Postes []
ct
MENTREE;Entrée du masque
define "MASQUE MLISTE :Postes [[] [ct ]]
define "ENTREE ELISTE :Postes :Pliste !
[[] [MASQUE make "LA []]]
```

```
end

to MENTREE
COMMANDE first cursor last cursor
MPOS if :AC = 252 [stop];Touche CLR
MENTREE
end

to COMMANDE :COL :Z
type char 7;signal sonore
setcursor [0 24]
type [Champ suivant - Commencer entree par] char 94
label "debut setcursor se :COL :Z
make "C rc make "AC ascii :C
if :AC = 240 [if :Z > 0 [make "Z :Z - 1]]
if :AC = 242 [make "Z :Z + 1]
if :AC = 250 [if :COL <39 [make "COL :COL + 1]]
if :AC = 254 [if :COL <0 [make "COL :COL - 1]]
if :AC = 243 [make "Z :Z + 1 make "COL 0]
if :AC = 94 [stop]
go "debut
end

to MPOS
make "CP cursor type char 7
setcursor [0 24]
(type [Entrer champ] char 94 [continuer, CLR: !
interruption])
setcursor CP
make "Postes lput se :CP MOT :Postes
make "Pliste lput cursor :Pliste
end

to MOT
make "W "
label "caractere make "C rc make "AC ascii :C
if :AC = 248 [make "W bl :W CGAUCHE]
if or :AC = 94 :AC = 252 [op :W]
if and :AC > 31 :AC < 125 [type :C make "W word :W :C]
go "caractere
```

```
end

to CGAUCHE;déplacer le curseur vers la gauche
type char 8;caractère de commande sur le CPC
end
```

La procédure MENTREE produit, après l'entrée de l'utilisateur, deux listes, Postes et Pliste.

Les éléments de Postes sont des listes de trois mots qui représentent chaque fois un champ d'entrée. Les deux premiers éléments indiquent la position du champ sur l'écran, le troisième élément le texte du formulaire correspondant. Cette liste contient les entrées nécessaires pour la définition de la procédure MASQUE.

La liste Pliste contient l'indication de la position où s'achève le texte de formulaire du champ et où commence le véritable champ d'entrée. Pliste se compose donc de listes de deux éléments. Cette indication est utilisée par la procédure ENTREE.

Exemple:

```
[0 2 Ville:] [6 2]
```

La première liste fait partie de Postes, la seconde figure dans l'emplacement correspondant de Pliste. Ainsi sera produit au début de la troisième ligne de l'écran (la première ligne porte le numéro 0) le texte Ville:, le curseur étant ensuite amené sur la septième colonne, à la suite du texte. Comme le début du prochain champ d'entrée est connu, le texte du formulaire peut être entièrement protégé contre un effaçage intempestif par l'utilisateur.

La procédure d'entrée MENTREE appelle en premier lieu la procédure COMMANDE. COMMANDE permet de commander le curseur pendant l'entrée du masque. L'utilisateur doit donc amener tout d'abord le curseur dans l'emplacement de l'écran voulu puis commencer l'entrée du texte du formulaire. Le Postes actuel, c'est-à-dire le texte du champ appelé est alors entré dans la procédure MPOS.

Aussi bien dans COMMANDE que dans MPOS, une ligne d'explication

est sortie dans le bas de l'écran. Le début et la fin du texte du formulaire sont marqués par un caractère spécial affiché par l'utilisateur. Nous avons choisi ici la flèche vers la gauche.

COMMANDE comporte toute une série de lignes if du type

```
if :AC = 242 [....]
```

Le nom AC contient le code ASCII du caractère entré. Avec les significations suivantes:

240: Curseur vers le haut
242: Curseur vers le bas
250: Curseur vers la droite
254: Curseur vers la gauche
243: Touche RETURN
94: Flèche vers le haut
248: Touche DEL (utilisée dans MOT)
252: Touche CLR (utilisée dans MOT)

On doit utiliser ici le code ASCII car les touches de commande ne peuvent être identifiées de la même façon que les caractères normaux.

Lorsque le curseur doit être commandé par le programme, une première possibilité consiste à appeler avec setcursor et cursor la position voulue sur l'écran:

```
setcursor se (first cursor) - 1 last cursor
```

déplacera par exemple le curseur d'une colonne vers la gauche. (Notez bien que les parenthèses sont indispensables ici. Sur le CPC, le curseur peut également être commandé en sortant les codes ASCII 8 à 11, comme nous l'avons fait dans la procédure CGAUCHE. Cette solution très proche de la machine n'est bien sûr pas tellement en harmonie avec la philosophie du LOGO. CGAUCHE fonctionne d'autre part également sur le PCW mais le PCW dispose en outre pour la gestion de l'écran de toute une série de ce qu'on appelle les séquences Escape (voir l'annexe 3 de votre manuel d'utilisation).

```
to MLISTE :Postes :ml
if emptyp :Postes [op :ml]
make "l first :Postes
make "l (list "setcursor piece 1 2 :l !
"type ( list item 3 :l ))
op MLISTE bf :Postes lput :l :ml
end

to ELISTE :Postes :Pliste :ml
if emptyp :Postes [op lput [op :LA] :ml]
(local "l "ms "li) make "l first :Postes make "li first !
:Pliste
make "Postes bf :Postes
if emptyp :Postes !
[make "ms 40] [make "ms first first :Postes]
if :ms > 0 [make "ms :ms - 1] [make "ms 39]
make "l ( se "make ""la "lput "IN :li :ms "" ":la)
op ELISTE :Postes bf :Pliste lput :l :ml
end
```

MLISTE et ELISTE forment, à partir des deux listes Postes et Pliste qui ont été produites lors de l'entrée du masque écran, les listes qui représenteront le texte de programme des procédures MASQUE et ENTREE souhaitées comme résultat final. Si des mots LOGO doivent être intégrés dans le texte de programme, ceux-ci doivent ici être précédés de guillemets.

C'est ainsi par exemple que le mot LOGO setcursor de la quatrième ligne de MLISTE devra être inséré dans la ligne correspondante du texte du programme MASQUE. S'il n'y avait pas de guillemets ici, la procédure LOGO serait exécutée immédiatement, ce qui n'est pas le but recherché. Si des guillemets doivent être utilisés dans le texte du programme à produire, les guillemets doivent eux-mêmes être précédés de guillemets, comme par exemple dans la septième ligne de ELISTE.

Dans ELISTE, la procédure IN est également prévue dans le texte de programme de ENTREE. Il s'agit ici de la procédure présentée à la leçon 10 pour l'entrée des données dans le champ d'entrée proprement dit. La voici à nouveau avec la procédure ADRESSES

qu'elle utilise également.

```
to ADRESSES :liste
local "personne
label "entree make "personne ENTREE
if first :personne = "* [op :liste]
make "liste lput :personne :liste
go "entree
end

to IN :S :Z :M :W
setcursor se :S :Z type char 32
setcursor se :S :Z
(local "c "ac)
label "touche make "c rc make "ac ascii :c
if :ac = 248 [op IN :S - 1 :Z :M bl :W]
if :ac = 243 [op :W]
if and :ac > 31 :ac < 123 !
[make "W word :W :c type :c] [go "touche]
if :S = :M [op :W]
op IN :S + 1 :Z :M :W
end
```

ADRESSES et IN ne font pas partie du paquet de programmes que nous vous présentons ici. ADRESSES permet plutôt d'utiliser les procédures MASQUE et ENTREE produites pour l'entrée de données personnelles.

Le paquet de programmes est mis en oeuvre avec

```
GENERATEURDEMASQUE
```

L'écran est alors vidé, le curseur clignote dans l'angle supérieur gauche. Le masque d'entrée est ensuite entré conformément aux instructions données dans la dernière ligne de l'écran. Une fois le texte du formulaire entré, l'entrée peut être interrompue avec la touche CLR. Le LOGO continue alors son travail pour créer les procédures MASQUE et ENTREE.

Les deux procédures peuvent être stockées sur disquette pour une

utilisation future. Il convient cependant de supprimer auparavant tout ce qui est inutile. Il s'agit entre autres de tous les noms qui peuvent être supprimés avec

```
ern glist ".APV
```

Toutes les procédures du paquet de programmes peuvent également être supprimées avec er.

La suppression des procédures inutiles peut être aussi programmée dans le programme GENERATEURDEMASQUE lui-même. Il faut alors lui ajouter les lignes suivantes:

```
...
ern glist ".APV
er [COMMANDE MPOS ELISTE MLISTE CGAUCHE]
er [MENTREE MOT GENERATEURDEMASQUE]
end
```

Avec cette modification ne subsisteront plus, après appel du programme, que les procédures produites MASQUE et ENTREE. Cette version ne doit bien sûr être mise en oeuvre qu'après que le paquet de programmes ait été stocké au moins une fois sur la disquette.

Après entrée d'un masque écran simple pour la saisie des nom, prénom et ville, le programme produira par exemple deux procédures telles que celles-ci:

```
to MASQUE
ct
setcursor [0 0] type [Nom:]
setcursor [22 0] type [Prenom:]
setcursor [0 3] type [Ville:]
end

to ENTREE
MASQUE make "la []
make "la lput IN 5 0 21 "" :la
```

```
make "la lput IN 30 0 39 "" :la
make "la lput IN 8 3 39 "" :la
op :la
end
```

Leçon 18: Gestion de fichier sous LOGO

Programmes:
Extension de l'entrée de tableau, STOCKER (SAVE avec remplacement et fichier BAK), DESCRIPTION (description d'un objet), DONNEES (saisie de données avec listes de propriétés)

18.1 Gestion de fichier

Presque tout le monde a aujourd'hui des fichiers à gérer, ne serait-ce par exemple, pour une personne privé, qu'un agenda téléphonique. Un agenda téléphonique est constitué d'une série d'entrées dont chacune comprend au moins le nom et le numéro de téléphone d'un correspondant. Mais d'autres renseignements tels que l'adresse, une différenciation entre privé et professionnel peuvent être intéressants.

Lorsqu'on possède beaucoup de livres, disques ou programmes informatiques, on souhaitera pouvoir en faire un fichier. Outre l'objectif général de disposer d'une documentation écrite, un tel fichier permet également de pouvoir accéder de manière sélective et délibérée à des objets quelconques. Dans une bibliothèque, on souhaitera par exemple pouvoir faire rechercher et sortir par ordre alphabétique une liste de tous les titres se rapportant à un domaine particulier.

Il ne s'agira cependant pas seulement, en général, de pouvoir accéder aux informations déjà disponibles. Il est tout aussi important de pouvoir en permanence compléter et modifier facilement le contenu du fichier.

Bien que la technique classique des bureaux ait depuis toujours développé les outils nécessaires à cet effet, l'ordinateur s'est

vite révélé convenir parfaitement à ce types de problèmes. On ne va cependant certainement pas placer un ordinateur personnel à côté de chaque téléphone mais on verra sans doute bientôt des ordinateurs personnels sur les bureaux de tous les responsables. Il n'est donc pas étonnant que l'on trouve déjà sur le marché de nombreux et puissants programmes de gestion de fichier.

Dans ce chapitre, nous vous présenterons un paquet de programmes LOGO pour la gestion de fichier qui n'a bien sûr pas pour ambition de concurrencer les logiciels commerciaux. Il s'agit bien plutôt d'analyser les actions de base qui apparaissent dans ce domaine et de les convertir en procédure pour être employées sous LOGO dans des programmes plus complexes.

Il faut en premier lieu choisir pour les fichiers une structure de données adaptée. Sous DR LOGO, on pense immédiatement aux listes de propriété. Une fiche du fichier sera représentée par un objet doté de certaines propriétés. Chaque renseignement porté sur la fiche du fichier sera une propriété, chaque entrée sera la valeur correspondante de cette propriété. Les fiches appartenant à un fichier donné devront avoir un nom commun et elles devront être numérotées. Sous LOGO, il est possible de construire des noms à partir d'éléments différents. C'est pourquoi les objets seront dotés d'un nom tel que Livre1, Livre2, etc...

18.2 Le menu de gestion

Le nom de fichier marque sera un nom global attendu pour être sauvegardé avec le fichier lors de sa sauvegarde sur disquette. L'appel pourra par exemple se faire avec la ligne suivante:

```
make "marque "Livre GESTION
```

Le menu suivant sera ensuite produit sur l'écran:

Creer nouveau fichier	1
Renumeroter	2
Autres enregistrements	3
Traiter enregistrements	4

Trier	5
Stocker	6
Rechercher	7

Après sélection, à l'aide des chiffres d'une des actions proposées, on appellera chaque fois une des procédures prévues à cet effet.

```
to GESTION
ct
(pr [Nom de fichier:] :marque) setcursor [0 4]
type [Creer nouveau fichier] setcursor [25 4] pr "1 pr "
type [Renumeroter] setcursor [25 6] pr "2 pr "
type [Autres enregistrements] setcursor [25 8] pr "3 pr "
type [Traiter enregistrements] setcursor [25 10] pr "4 !
pr "
type [Trier] setcursor [25 12] pr "5 pr "
type [Stocker] setcursor [25 14] pr "6 pr "
type [Rechercher] setcursor [25 16] pr "7 pr "
catch "error [run (list word "a rc)]
type [Frapper une touche pour continuer!] pr rc GESTION
end
```

Une fois le menu sorti, le nom de la procédure correspondant à chaque point du menu, c'est-à-dire A1, A2 ... A7, est construit avec word "a rc et le programme sélectionné est alors mis en oeuvre avec le mot LOGO run. L'appel de procédure se fait ici sous couvert d'un catch "error qui entraîne un saut à l'avant-dernière ligne du programme GESTION en cas d'apparition d'une erreur.

Si les données souhaitées n'ont par exemple pas été trouvées, ce qui peut assez facilement se produire à la suite d'une entrée erronée, le message "Frapper une touche pour continuer!" apparaît à l'écran.

Le menu représenté montre les possibilités du paquet de programmes. Nous allons maintenant les examiner séparément l'une après l'autre.

18.3 Création d'un fichier

Creer nouveau fichier

Le premier point du menu permet d'écrire les propriétés, c'est-à-dire les rubriques de renseignements souhaités, sur une fiche, pour le nouveau fichier. Les données dont on dispose pourront ensuite être saisies. Le programme appelé s'appelle A1 et il appelle de son côté les programmes DONNEES, SAISIR, ENREGISTREMENT et PROTOTYPE.

```
to A1
ct (pr :marque [Creer nouveau fichier])
DONNEES :marque
end

to DONNEES :marque
(pr [Saisie des donnees avec la marque:] :marque)
pr [Veuillez enumerer les rubriques souhaitees]
catch "fin [SAISIR 1 PROTOTYPE rl word :marque "1]
end

to SAISIR :i :proto
local "nom make "nom word :marque :i
ct (pr [Enregistrement:] :nom
ENREGISTREMENT :nom :proto
make "imax :i;Numero d'enregistrement le plus eleve
SAISIR :i+1 :proto
end

to ENREGISTREMENT :nom :propre
(local "e "w)
if emptyp :propre [stop]
make "e last :propre make "propre bl :propre
(type last :propre ": :e) make "w rq
if numberp :w [make "w (+ :w)]
if :w = "eof [throw "fin]
if emptyp :w [make "w :e]
pprop :nom last :propre :w
ENREGISTREMENT :nom bl :propre
```

```
end

to PROTOTYPE :pro :nom
if emptyp :pro [op plist :nom]
pprop :nom first :pro "
op PROTOTYPE bf :pro :nom
end
```

Le programme A1 appelle immédiatement la procédure DONNEES; on ne passe par le détour de A1 que pour avoir le nom explicatif DONNEES. Dans DONNEES, l'utilisateur est prié d'énumérer les renseignements souhaités, ce qui n'exclut toutefois pas que d'autres rubriques puissent être ajoutées au cours d'un traitement ultérieur. L'appel de la procédure SAISIR est alors effectué avec catch "fin. le catch "fin a l'avantage de permettre une interruption prématurée des autres procédures appelées par la suite.

Dans le programme SAISIR, la procédure ENREGISTREMENT est appelée de manière récursive, le numéro de code étant augmenté chaque fois. La saisie des données se termine lorsqu'un "eof (pour end of file) est entré pour la première propriété d'un objet. Le test correspondant est effectué dans la procédure ENREGISTREMENT et on saute alors directement à DONNEES avec throw "fin.

La première entrée pour ENREGISTREMENT est le nom de l'enregistrement, Livre1 ou Disque5 par exemple, la seconde est une liste de propriétés avec leurs valeurs respectives. Lors de la création d'un nouveau fichier, aucune entrée n'est disponible et c'est pourquoi la liste nécessaire doit être d'abord produite à l'aide du programme PROTOTYPE lors de l'appel de SAISIR. La méthode utilisée permet d'utiliser les procédures SAISIR et ENREGISTREMENT également dans d'autres points du menu qui servent à l'entretien du fichier.

La liste des rubriques est maintenant traitée par récursion dans le programme ENREGISTREMENT. S'il y a déjà une entrée, celle-ci apparaîtra à l'écran. En l'absence d'entrée de l'utilisateur, c'est l'entrée présentée qui sera reprise après que la touche RETURN ait été enfoncée. Ce procédé a également été choisi en pensant à la

perspective de l'entretien du fichier, même si la disposition de l'écran obtenue n'est pas encore optimale.

On teste dans ENREGISTREMENT si c'est un nombre qui a été entré. Dans ce cas, la valeur sera redéfinie avec make "w (+ :w). Cela peut sembler curieux de prime abord puisque cela ne change rien à la valeur numérique obtenue. Cette instruction est utilisée parce que nous avons observé que, si l'on ne prend pas de mesure de ce type, c'est l'ordre "alphabétique" qui sera utilisé pour toute comparaison de valeurs de propriété avec > ou <. Or cela entraîne que 1000 sera considéré comme inférieur à 2. L'astuce utilisée a donc pour but de rappeler à DR LOGO la signification numérique de la propriété considérée.

Le programme PROTOTYPE affecte enfin à l'objet transmis sous nom les rubriques énumérées par l'utilisateur. On prendra ici comme valeur un mot vide. L'objet possède donc ensuite, sur le plan formel, les propriétés voulues et la véritable entrée des valeurs consistera en quelque sorte à modifier les valeurs de propriétés.

18.4 Réorganisation d'un fichier

Renumérotation

Si on éliminer certains enregistrements du fichier, par exemple Livre3 du fichier Livre, un vide apparaît dans la numérotation. Pour combler ce vide, tous les enregistrements suivants doivent être ramenés d'une position en avant mais cela ne peut être fait directement avec les listes de propriété. Il faut en fait recopier intégralement le texte des fiches suivantes sur la fiche chaque fois précédente et cela représente un gros travail.

C'est pourquoi nous nous résignerons dans un premier temps à voir des vides apparaître ici ou là. Le second point du menu nous permettra donc, lorsque nous le souhaiterons de faire renuméroter le fichier tout entier de façon à éliminer tous les vides apparus.

```
to A2
ct (pr :marque [Renumeroter]
```

```
pr [Limite superieure:]
make "imax NUMERO 1
end

to NUMERO :i
if :i > ;imax [op :imax]
if not emptyp plist word :marque :i [op NUMERO :i+1]
local "ne make "ne SUIV :i+1
if emptyp :ne [op :i - 1]
DEPLACE word :marque :i :ne
op NUMERO :i + 1
end

to SUIV :i
if :i > :imax [op "]
local "j make "j word :marque :i
if emptyp plist :j [op SUIV :i+1] [op :j]
end

to DEPLACE :h :k
local "sl make "sl plist :k
if emptyp :sl [stop]
pprop :h last bl :sl last :sl
remprop :k last bl :sl
DEPLACE :h :k
end
```

Les enregistrements existants peuvent avoir des numéros compris entre un et :imax. La gestion du numéro le plus élevé s'effectue avec la variable globale imax qui sera également stockée sur la disquette lors de la sauvegarde.

On détermine avec plist si une enregistrement existe; plist fournit en effet une liste vide comme résultat si l'objet ne possède aucune propriété. La procédure SUIF recherche l'enregistrement suivant alors que DEPLACE transfère enfin les propriétés sur l'enregistrement libre de numéro moins élevé.

18.5 Extension et entretien du fichier

Autres enregistrements

La tâche à accomplir dans ce point du menu est simple: il s'agit de permettre d'ajouter des enregistrements supplémentaires au fichier.

```
to A3
ct (pr [Autres enregistrements pour:] :marque
(pr [en commencant au numero:] :imax+1)
catch "fin [SAISIR :imax+1 plist word :marque "1]
end
```

Les enregistrements supplémentaires sont ajoutés de la même façon que dans le premier point du menu avec les procédures SAISIR et ENREGISTREMENT. Les propriétés voulues sont reprises du premier enregistrement qui est utilisé comme prototype.

Traitement des enregistrements

Pour entretenir un fichier, les possibilités suivantes ont été prévues:

- suppression d'enregistrements isolés,
- modification de valeurs de propriétés existantes,
- ajout de propriétés supplémentaires.

```
to A4
TRAITER 1
end

to TRAITER :i
local "nom
if :i > :imax [stop]
make "nom word :marque :i
ct (pr [Enregistrement:] :nom)
ENROUT plist :nom setcursor [0 15]
pr [Prochain enregistrement] setcursor [23 15] pr "1
pr [Supprimer enregistrement] setcursor [23 16] pr "2
```

```
pr [Modifier propriétés] setcursor [23 17] pr "3
pr [Ajouter propriétés] setcursor [23 18] pr "4
run item rc [[] [SUPPRIME :nom] [ENREGISTREMENT :nom plist :nom] !
[AJOUTER :nom]]
TRAITER :i+1
end

to ENROUT :eliste
(local "e "w)
if emptyp :eliste [stop]
make "e last :eliste make "eliste bl :eliste
(pr last :eliste ": :e)
ENROUT bl :eliste
end

to SUPPRIME :nom
local "e make "e plist :nom if emptyp :e [stop]
remprop :nom first :e
SUPPRIME :nom
end

to AJOUTER :nom
(local "e "w)
type [Propriété voulue:]
make "e rq if emptyp :e [stop]
type [Valeur:] make "w rq
if numberp :w [make "w (+:w)]
pprop :nom :e :w
AJOUTER :nom
end
```

Au début de la procédure TRAITER, l'enregistrement (numéro i) est sorti. On utilise à cet effet la procédure ENROUT qui fonctionne comme ENREGISTREMENT si ce n'est qu'elle n'attend elle-même aucune entrée. Un sous-menu, c'est-à-dire une sélection supplémentaire, est alors proposé. La technique du menu est utilisée comme dans le programme principal GESTION. L'utilisateur entre le chiffre de code voulu.

La commande des différentes possibilités ne se fait cependant pas

ici de la même manière que dans le programme principal. On ne construit pas de noms de procédure mais l'action à déclencher est tirée d'une liste de quatre éléments avec item rc. Elle est ensuite exécutée avec run. Si on doit passer à l'enregistrement suivant, il n'y a rien d'autre à faire et c'est pourquoi le premier élément de la liste des actions à exécuter est une liste vide.

Pour supprimer un enregistrement, on utilise la procédure SUPPRIME qui élimine tous les propriétés de l'objet entré. Pour modifier des propriétés, on appelle le programme ENREGISTREMENT que nous avons déjà utilisé pour les points un et trois du menu principal.

Le dernier point du sous-menu appelle la procédure AJOUTER qui permet d'ajouter des propriétés supplémentaires à l'enregistrement actuellement traité. AJOUTER est également construit sur le modèle de ENREGISTREMENT si ce n'est que cette procédure ne se termine que lorsque l'utilisateur répond sans entrée, en appuyant directement sur la touche RETURN, à la demande de la propriété voulue.

18.6 Sortie triée

Nous avons dévelppé à la leçon 16 le programme de tri LINSORT, c'est-à-dire tri par insertion linéaire. Nous allons maintenant pouvoir l'utiliser pour notre gestion de fichier. On voudra par exemple sortir tous les livres du fichier Livre triés par ordre alphabétique de nom d'auteur ou d'après l'ordre de parution ou encore en fonction des prix éventuellement obtenus.

```
to A5
ct (pr [Sortie triee pour: ] :marque)
pr [Critere de tri: ]
local "critere make "critere rq
ct (pr [Sortie d'apres] :critere
TRIA TRIE :critere
end

to TRIA :d
if emptyp :d [stop]
```

```
ENROUT plist first :d
TRIA bf :d
end

to TRIE :critere
local "fichier make "fichier glist :critere
op LINSORT :fichier
end

to LINSORT :fichier
op LIN (list first :fichier) bf :fichier
end
to LIN :v :h
if emptyp :h [op :v]
op LIN IN :v first :h bf :h
end
to IN :l :e
if emptyp :l [op (list :e)]
if gprop first :l :critere < gprop :e :critere !
[op fput first :l IN bf :l :e] [op fput :e :l]
end
```

On demande tout d'abord le critère du tri qui doit naturellement être une propriété existant dans le fichier. Le critère est alors transmis au programme TRIE qui livrera comme résultat la liste des enregistrements triés.

Les enregistrements sont ensuite sortis sur l'écran avec la routine ENROUT que nous avons déjà utilisée.

Le program TRIE effectue tout d'abord des préparatifs nécessaires pour le tri en affectant, avec le mot LOGO glist, au nom fichier la liste de tous les objets auxquels s'applique la propriété entrée comme critère de tri. Le programme de tri LINSORT déjà décrit accède aux enregistrements par l'intermédiaire de cette liste.

Nous vous avons à nouveau donné le texte des trois procédures LINSORT, LIN, IN. Notez bien que l'accès à travers la clé a été intégré directement dans la troisième ligne de IN, avec gprop first :l :critere, sans faire le détour par un programme CLE.

Le tri se traduit finalement par la création, comme résultat, d'une liste de type [Livre7 Livre1 Livre3 ...]. Les propriétés des objets sont alors réunies avec plist dans l'odre Livre7, Livre1, Livre3, ... puis sorties par la procédure ENROUT.

18.7 Recherche et sauvegarde

```
to A6
pr [Supprimer procedures avant la sauvegarde? o/n]
if lc rc ="o [er glist ".DEF]
catch "error [save :marque]
if emptyp error [stop]
(pr [Fichier] :marque [existe deja])
pr [Supprimer ancien fichier? o/n]
if lc rc ="o [erasefile :marque save :marque] !
[pr [Le fichier n'a pas ete sauvegarde!]]
end
```

Il est tentant d'utiliser le nom disponible sous le nom global marque également comme nom de fichier lors de la sauvegarde. Mais l'instruction save transfère systématiquement sur la disquette tout le contenu de la mémoire de travail. (Il existe cependant aussi des versions du LOGO qui permettent de regrouper certaines parties de la mémoire de travail dans ce qu'on appelle des paquets qui pourront ensuite être gérés séparément.) Toutes les parties du programme seront donc notamment sauvegardées sur la disquette. Cela présente l'avantage de permettre qu'une opération unique de chargement suffise pour la gestion de fichier.

On peut cependant également supprimer toutes les procédures avant de sauvegardere le fichier de façon à ce que ce fichier occupe moins de place sur la disquette. La procédure A6 vous offre cette possibilité. Toutes les procédures seront supprimées avec l'instruction er glist ".DEF, ce qui inclut la procédure A6 qui sera alors mise en oeuvre. Avec cette variante, les procédures devront donc à nouveau être chargées pour poursuivre la gestion de fichier. Avec cette méthode, on peut déclencher le chargement automatique des programmes à l'aide du nom de fichier standard STARTUP sur la disquette système.

Si le nom de fichier existe déjà lors de la sauvegarde, ce qui est d'ailleurs tout à fait normal après que le fichier aura été sauvegardé une première fois lors de sa création, on vous propose de supprimer le fichier déjà existant. Si l'instruction changef fonctionnait correctement, il serait toutefois préférable de l'utiliser ici pour que l'ancien fichier ne soit pas supprimé mais que son nom soit simplement changé.

Rechercher

Le point du menu que nous étudierons en dernier est souvent l'action la plus utilisée dans les programmes de gestion de fichier. L'organisation efficace de l'accès sélectif aux données est un problème important et difficile à la fois en informatique. Nous vous présentons ici simplement une comparaison systématique des enregistrements à un critère de recherche.

```
to A7
(local "cherche "compare)
ct pr [Critere de recherche: ] make "cherche rq
pr " pr [Comparaison voulue:]
make "compare rl
CHERCHER 1
end

to CHERCHER :i
if :i > :imax [stop]
(local "nom "valeur)
make "nom word :marque :i
make "valeur gprop :nom :cherche
if emptyp :valeur [go "suite]
if run se ":valeur :compare [ENROUT plist :nom]
label "suite CHERCHER :i+1
end
```

Il faut entrer comme critère de recherche une propriété existante. La procédure CHERCHER examinera tous les enregistrements pour voir si la valeur correspondant à cette propriété remplit la condition posée par l'utilisateur.

On pourrait par exemple prendre comme critère de recherche dans le fichier Livre la date de parution et comme condition "> 1980". Dans l'antépénultième ligne de CHERCHER, la condition est testée avec

```
run se ":valeur :compare
```

Dans l'exemple considéré, c'est par exemple le nombre 1986 qui pourrait être stocké sous le nom :valeur. La liste

```
[:valeur > 1980]
```

sera alors formée avec se ":valeur :compare puis exécutée avec run pour donner une valeur de vérité, c'est-à-dire TRUE ou FALSE. Si la condition est remplie, l'enregistrement sera sorti.

La méthode décrite ici est déjà assez souple pour permettre d'accéder aux enregistrements voulus. Si vous voulez par exemple rechercher tous les titres d'un auteur donné, vous entrerez "Auteur" comme critère de recherche et ensuite par exemple "= "Eluard comme comparaison. (Notez bien que les guillemets précédant le nom sont indispensables.)

Si vous voulez utiliser des critères de recherche compliqués, le mieux est de remplacer l'antépénultième ligne de CHERCHER par

```
if run :compare [ENROUT plist :nom]
```

La valeur de la propriété recherchée devra être intégrée explicitement, dans cette variante, dans le critère de comparaison. Avec l'exemple de l'année de parution, on devrait par exemple entrer:

```
:valeur > 1980 au lieu de > 1980
```

Mais l'avantage est que vous pouvez maintenant utiliser également des opérations logiques pour définir le critère de recherche. Pour une recherche d'après le nom de l'auteur par exemple, vous pourrez entrer:

```
or :valeur = "Eluard :valeur = "Einstein
```

Vous obtiendrez ainsi une liste des ouvrages de ces deux auteurs. Pour une recherche d'après l'année de parution, vous pourriez définir une période de recherche avec

```
and :valeur > 1920 :valeur < 1950
```

Leçon 19: Couleur et son

Nouveaux éléments de vocabulaire:
setbg, setpc, setpal, pal, sound, env, ent, release

Programmes:
CHAINE, CHANGECOULEUR, NUM, PERIODE

La dernière leçon sera consacrée à la programmation des couleurs ainsi qu'à la programmation de générateurs de son. Les couleurs ne présentent bien sûr d'intérêt que pour les possesseurs d'ordinateurs CPC avec moniteur couleur. Les générateurs de son existent sur tous les modèles CPC mais pas sur le PCW AMSTRAD.

19.1 Les couleurs de base rouge, vert et bleu

L'image couleur est produite sur les moniteurs couleur, comme sur les téléviseurs couleur à partir des trois couleurs de base rouge, vert et bleu (RVB), l'impression de couleur effective résultant de la superposition des trois images que les physiciens désignent également sous le terme de mélange de couleur additif.

Chaque couleur de base possède son propre "canon" de couleur qui est commandé séparément. En modifiant l'intensité, c'est-à-dire la clarté de chacune des trois couleurs de base sur un point d'image donné, on peut produire toutes les nuances de couleur possibles. Sur une image informatique, l'intensité ne peut cependant varier que par étapes déterminées et séparées. L'intensité est en effet commandée de façon digitale.

Sur le CPC AMSTRAD, vous ne disposez pour chaque couleur de base que de trois niveaux de clarté

0, 1 ou 2

et vous pouvez par conséquent produire

3 * 3 * 3 = 27

tons de couleur différents. C'est pourquoi vous trouvez sur la droite du boitier du lecteur de disquette intégré une table comportant les codes de couleur 0 à 26. Ces numéros de couleur ne sont toutefois pas utilisés sous cette forme en LOGO.

En réalité, on ne peut pas utiliser simultanément les 27 couleurs possibles. Vous disposez toujours en LOGO d'une palette de quatre pots de peinture à un moment donné. Mais chacun de ces pots de peinture peut contenir l'une quelconque des 27 nuances de couleur disponibles.

Cela semble peut-être un peu compliqué mais cela recouvre une réalité très simple: vous disposez de trois crayons de couleur différent avec une quatrième couleur pour la feuille de dessin, c'est-à-dire pour le fond de l'écran. La couleur effectivement visible sur l'écran sera pour chaque crayon une des 27 combinaisons possibles des trois couleurs de base rouge-vert-bleu. Cette sélection peut être modifiée à tout moment, y compris à l'intérieur d'un programme, mais, et c'est important à noter, tous les points qui ont déjà été dessinés sur l'écran avec le crayon de couleur dont vous modifiez maintenant la couleur changeront automatiquement de couleur.

19.2 Sélection des couleurs

Sélection des crayons de couleur

Sur la fenêtre de texte, vous n'avez aucun choix. Le fond correspondra toujours au pot de peinture de numéro zéro et le crayon au pot de numéro un.

Dans la fenêtre graphique, la couleur du fond peut être fixée avec le mot LOGO setbg et celle du crayon à dessin, qui est aussi celle de la tortue, avec l'instruction setpc.

```
setbg 1 setpc 0
```

Une modification du numéro de fond ne se remarquera qu'après que la fenêtre graphique ait été vidée avec cs ou clean.

Les fonctions st (screen facts) et tf (turtle facts) vous permettent d'obtenir les valeurs actuelles des pots de peinture.

```
sf tf
```

Réponse: par exemple [1 SS 5 WINDOW] [0 0 0 PD 0 TRUE]

Dans la liste de réponse de sf, le premier élément indique la couleur du fond, dans celle de tf l'avant-dernier élément indique la couleur du crayon à dessin.

Les valeurs autorisées sont chaque fois 0, 1, 2, 3 sur les machines dotées de moniteur couleur. Sur les écrans monochromes, vous ne disposez que des couleurs 0 ou 1.

Sélection de la couleur de l'écran

La couleur correspondant à chaque pot de peinture peut être fixée par indication de l'intensité sélectionnée pour chacune des trois couleurs de base.

```
setbg 0 setpal 0 [2 0 0]
```

On ne sélectionnera ici dans le pot de peinture que la couleur rouge avec une intensité de 2 et les deux autres couleurs manquent. L'écran montrera de ce fait un fond rouge. En tout cas cela devrait faire rougir le fond de la fenêtre de texte.

setpal a deux entrées: le numéro de couleur appelé et une liste composée de trois éléments qui indique la combinaison de couleurs correspondante. Essayez:

```
repeat 100 !
[setpal 0 (se random 3 random 3 random 3) wait 10]
```

La couleur du fond de l'écran sera ainsi modifiée en permanence par tirage au sort. Si vous en profitez pour observer l'impression de couleur produite par un dessin ou un texte lorsque la couleur du fond change, vous remarquerez qu'un changement de couleur peut aussi être obtenu, sans changer de pot de peinture ni la nuance de couleur affectée à ce pot de peinture, simplement en modifiant le contraste de couleur par rapport au fond de l'écran. Cela peut entraîner des surprises en ce qui concerne les effets de couleurs recherchés, notamment lorsqu'on a affaire à des lignes fines. Pour obtenir l'organisation des couleurs recherchée, il convient donc souvent de faire de nombreux essais avant de parvenir à une forme définitive parfaitement satisfaisante.

setpal vous permet naturellement également de faire des expériences élémentaires de mélange additif de couleurs.

```
ts cs setpal 0 [2 0 0] wait 50 setpal 0 [0 2 0] wait 50
setpal 0 [2 2 0]
```

Vous devriez voir apparaître sur le fond de l'écran d'abord les deux couleurs rouge et vert puis l'addition des deux, c'est-à-dire le jaune.

```
pal 0
```

Réponse: [2 2 0]

Vous pouvez ainsi effectuer également un échange de couleurs entre différents pots de peinture.

```
setpal 0 pal 1
```

La couleur du pot 1 sera versée dans le pot de numéro 0, ce qui rendra le texte de la fenêtre de texte invisible puisque l'écriture et le fond auront maintenant la même couleur.

Effets de mouvement

Un changement de la palette de couleurs permet de rendre visibles

ou invisibles certaines parties d'un dessin ou de modifier considérablement l'impression de couleur se dégageant de certains dessins. Avec un changement rapide, on peut ainsi faire naître l'impression d'un mouvement sans que le dessin véritable ait été réellement modifié.

L'exemple suivant vous donnera un aperçu des effets qu'on peut ainsi obtenir.

```
to Q :l;carre rempli
pd
repeat 5 [seth 0 fd :l rt 90 fd 2 lt 90 bk :l rt 90 fd 2]
pu fd :l
end

to CHAINE :l
setpal 0 [2 2 2]
setpal 3 pal 0
setpal 2 pal 0
setpal 1 pal [2 0 0]
c pu setx -320 KE :l 1
end

to KE :l :i
if first tf > 320 [stop]
setpc :i Q :l
if i=3 [KE :l 1] [KE :l :i + 1]
end

to CHANGECOULEUR
setpal 1 [2 2 2]
setpal 2 [2 0 0]
wait 10
setpal 2 [2 2 2]
setpal 3 [2 0 0]
wait 10
setpal 3 [2 2 2]
setpal 1 [2 0 0]
wait 10
CHANGECOULEUR
```

```
end
```

Appel: fs cs CHAINE 20 CHANGECOULEUR

Q dessine des carrés pleins qui sont accolés horizontalement l'un à la suite de l'autre avec CHAINE. Un carré sur trois est d'abord rempli de rouge, entre eux s'intercalent deux carrés dont la couleur est celle de l'écran graphique (blanc en l'occurrence).

La procédure CHANGECOULEUR efface maintenant chaque fois les carrés coloriés et colore ensuite de rouge les carrés situés à leur droite. Ce changement périodique de couleur donne l'impression qu'une chaîne de carrés rouges court sur l'écran de gauche à droite.

Comme les cases carrées ont toutes la même forme, l'effet de mouvement n'est bien sûr pas très impressionnant. Essayez donc, par exemple, de dessiner avec des crayons de couleurs différents des phases successives du mouvement d'une figure puis de représenter ces différentes phases en mouvement périodique en utilisant la technique du programme CHANGECOULEUR! Cette technique permettrait naturellement des effets encore plus impressionnants si on pouvait représenter simultanément plus de couleurs différentes.

Pour vous donner encore une idée de jeu avec les couleurs, nous allons maintenant revenir au programme BASCULE de la leçon 14. La procédure CHAMP que nous y utilisions devra maintenant être modifiée comme suit pour jouer sur les couleurs.

```
to CHAMP :y
if :y < -180 [stop]
pu setx 320 sety :y seth 0 pd make "i 2
repeat 8 [setpc 1 par 120 2 12 setpc :i para 60 12 2 !
make "i vers :i]
pu setx 320 sety :y - :ns - :l seth 0 pd
repeat 8 [setpc 1 para 60 12 2 make "i vers i setpc 1 !
par 120 2 12]
CHAMP :y - :ns -:l -:l
end
to vers :i
```

```
op item :i [2 3 1]
end
to CHANGECOULEUR
make "k pal 0
setpal 0 pal 1 setpal 1 pal 2
setpal 2 pal 3 setpal 3 :k wait 30
CHANGECOULEUR
end
```

Après appel du programme BASCULE (leçon 14), CHANGECOULEUR permet de mettre en oeuvre un jeu de couleurs changeantes. Les couleurs peuvent être fixées au départ à votre guise.

19.3 Programmation des générateurs de son

Les ordinateurs CPC AMSTRAD disposent d'un générateur de son puissant à trois voix. Il ne nous est, pour diverses raisons, pas possible d'en expliquer en détail la programmation. Si nous voulions expliquer les possibilités de production des sons de façon suffisamment claire, il nous faudrait déjà un long chapitre. Nous vous conseillons donc plutôt, si ce sujet vous intéresses, d'étudier la section consacrée à la musique dans le chapitre 9 de votre manuel d'utilisation.

Vous y trouverez des instructions qui appartiennent en réalité au langage de programmation BASIC. Ces instructions ont cependant été reprises telles quelles, avec leur nom, dans DR LOGO. Il a simplement fallu réaliser la légère mais indispensable adaptation à la syntaxe du langage LOGO. La transposition ne s'est malheureusement pas faite sans quelques erreurs ou bévues.

L'instruction de base est sound:

```
sound [1 142 50]
```

Le premier élément de la liste d'entrée indique le canal sonore voulu, le second représente ce qu'on appelle la période de note qui est liée à la fréquence du son d'après la formule

```
Période de note = round 62500/fréquence
```

Dans l'exemple indiqué, il s'agit du la du diapason. Le troisième élément de la liste indique la durée de la note.

En BASIC, la même instruction aura la forme

```
SOUND 1,142,50,
```

Cette instruction est expliquée en détail dans la section consacrée à la musique du chapitre 9 de votre manuel d'utilisation.

Vous pouvez faire jouer la gamme chromatique de la façon suivante:

```
to NUM :note
if memberp :note [do do.diese re re.diese mi fa !
fa.diese sol sol.diese la la.diese si] [op item where !
[3822 3608 3405 3214 3034 2863 2703 2551 2408 2273 2145 !
2025] [op 0] end

to PERIODE :note :octave
op round (NUM :note)/item :octave [1 2 4 8 16 32 64 128]
end
```

La fonction PERIODE fournit la valeur de la période de note nécessaire pour l'instruction sound. La première entrée représente le nom de la note, la seconde l'octave. Avec la forme de PERIODE que nous vous présentons, le la du diapason sera produit avec

```
sound se 1 PERIODE "la 5
```

Les numéros d'octave vont ici de 1 à 8 mais vous pouvez bien sûr aisément modifier cette numérotation.

Les musiciens préféreront peut-être d'autre méthodes de désignation des notes, par exemple en pouvant indiquer également la hauteur de note absolue de façon symbolique. Si on veut transposer la notation musicale en un format aisément utilisable par l'ordinateur, on peut également procéder de telle façon que ne soient par exemple notés, en ce qui concerne l'indication de l'octave, que les passages d'une

octave à l'autre. Il serait donc intéressant à cet égard d'utiliser le traitement des listes sous LOGO pour développer des procédures auxiliaires pratiques qui permettront d'entrer les notes de façon aussi simple et aussi proche de la notation musicale traditionnelle que possible. Mais nous n'avons plus la place suffisante pour traiter ce sujet dans le cadre de cet ouvrage.

Nous n'avons jusqu'ici travaillé qu'avec un seul canal sonore, les deux autres sont commandés à travers les numéros 2 et 4. Effectivement, la première valeur en entrée pour sound est un nombre compris entre zéro et 255, ce qui correspond pour l'ordinateur, sur le plan interne, à un nombre binaire de 8 bits. Les différents bits sont interprétés séparément par l'instruction sound. Les trois derniers bits indiquent les générateurs de son appelés, 7 appelant donc les trois en même temps.

En ce qui concerne la signification des trois autres bits, qui commandent ce qu'on appelle la technique des rendez-vous et des files d'attente, nous devons une nouvelle fois vous renvoyer à votre manuel d'utilisation.

Outre la durée, vous pouvez faire également des indications facultatives sur le volume et ce qu'on appelle les courbes de volume et d'enveloppe ainsi que pour la production d'effets de bruits. Il vous faut alors indiquer des paramètres supplémentaires pour sound.

La courbe de volume vous permet de modifier par étapes le volume d'un son, c'est-à-dire normalement de le faire monter brusquement puis de le faire retomber plus progressivement. La courbe d'enveloppe vous permet en outre de modifier par étapes la période de note un peu comme on produit un vibrato sur les instruments à cordes. Ici aussi, nous ne pouvons que vous recommander de lire la section consacrée à la musique au chapitre 9 de votre manuel d'utilisation.

La définition de courbes se fait avec les mots LOGO ent et env qui correspondent très exactement aux instructions BASIC décrites dans votre manuel d'utilisation, si ce n'est que les valeurs en entrée ne doivent pas être séparées entre elles par des virgules mais être

réunies sous forme d'une liste. Il nous faut malheureusement constater ici que DR LOGO rejette tous les nombres négatifs comme valeurs de paramètres bien que les nombres négatifs soient également nécessaires au fonctionnement de ces instructions.

Dans la courbe de volume, on utilisera par exemple les nombres négatifs pour faire décroître le volume. Il y a toutefois une solution à ce problème puisque la valeur 15 a le même effet que -1, 14 que -2, etc...

Pour la courbe d'enveloppe, un numéro de courbe négatif permet d'obtenir une répétition périodique de la courbe. L'auteur du présent ouvrage n'est malheureusement pas parvenu à contourner le rejet des valeurs négatives par DR LOGO.

La dernière instruction à évoquer est release qui permet de libérer des canaux de son. Cette instruction fonctionne également en LOGO comme décrit dans le manuel d'utilisation pour l'instruction BASIC de même nom.

Achevé d'imprimer en novembre 1986
sur les presses de l'imprimerie Laballery
58500 Clamecy
Dépôt légal : novembre 1986
Numéro d'imprimeur : 610079

les livres Amstrad



TRUCS ET ASTUCES POUR L'AMSTRAD CPC (Tome 1)

C'est le livre que tout utilisateur d'un CPC doit posséder. De nombreux domaines sont couverts (graphismes, fenêtres, langage machine) et des super programmes sont inclus dans ce best-seller (gestion de fichiers, éditeur de textes et de sons...).
Réf. : ML 112
Prix : **149** FF



PROGRAMMES BASIC POUR LES CPC (Tome 2)

Alimentez votre CPC
Ce livre contient de supers programmes, notamment un désassembleur, un éditeur graphique, un éditeur de texte... Tous les programmes sont prêts à être tapés et abondamment commentés.
Réf. : ML 118
Prix : **129** FF

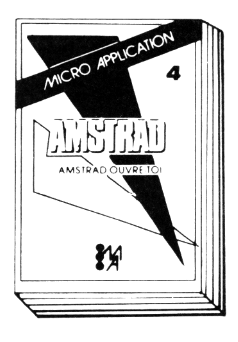
LE BASIC AU BOUT DES DOIGTS CPC (Tome 3)

Ce livre est une introduction complète et didactique au BASIC du micro-ordinateur AMSTRAD CPC 464. Il permet d'apprendre rapidement et facilement la programmation (instructions BASIC, analyses des problèmes, algorithmes complexes...).
Comprenant de nombreux exemples, ce livre vous assure un apprentissage simple et efficace du BASIC CPC.
Réf. : ML 119
Prix : **149** FF



AMSTRAD OUVRE-TOI (Tome 4)

Le bon départ avec le CPC 464 ! Ce livre vous apporte les principales informations sur l'utilisation, les possibilités de connexions du CPC 464 et les rudiments nécessaires pour développer vos propres programmes. C'est le livre idéal de tous ceux qui veulent pénétrer dans l'univers des micro-ordinateurs avec le CPC.
Réf. : ML 120
Prix : **99** FF





JEUX D'AVENTURES. COMMENT LES PROGRAMMER (Tome 5)

Voici la clé du monde de l'aventure. Ce livre fournit un système d'aventures complet, avec éditeur, interpréteur, routines utilitaires et fichiers de jeux. Ainsi qu'un générateur d'aventures pour programmer vous-mêmes facilement vos jeux d'aventures. Avec, bien sûr, des programmes tout prêts à être tapés.
Réf. : ML 121
Prix : **129** FF



LA BIBLE DU PROGRAMMEUR DE L'AMSTRAD CPC (Tome 6)

Tout, absolument tout sur le CPC 464. Ce livre est l'ouvrage de référence pour tous ceux qui veulent programmer en pro leur CPC. Organisation de la mémoire, le contrôleur vidéo, les interfaces, l'interpréteur et toute la ROM DESASSEMBLEE et COMMENTEE sont quelques-uns des thèmes de cet ouvrage de 700 pages.
Réf. : ML 122
Prix : **249** FF

les plus de Micro Application



LE LANGAGE MACHINE DE L'AMSTRAD CPC (Tome 7)

Ce livre est destiné à tous ceux qui désirent aller plus loin que le BASIC. Des bases de la programmation en assembleur à l'utilisation des routines système, tout est expliqué avec de nombreux exemples. Contient un programme assembleur, moniteur et désassembleur.

Réf. : ML 123
Prix : **129** FF



GRAPHISMES ET SONS DU CPC (Tome 8)

L'AMSTRAD CPC dispose de capacités graphiques et sonores exceptionnelles. Ce livre en montre l'utilisation à l'aide de nombreux programmes utilitaires.

Réf. : ML 124
Prix : **129** FF

PEEKS ET POKES DU CPC (Tome 9)

Comment exploiter à fond son CPC à partir du BASIC? C'est ce que vous révèle ce livre avec tout ce qu'il faut savoir sur les peeks, pokes et autres call... Vous saurez aussi comment protéger la mémoire, calculer en binaire... et tout cela très facilement. Un passage assuré et sans douleur du BASIC au puissant LANGAGE MACHINE.

Réf. : ML 126
Prix : **99** FF



LIVRE DU LECTEUR DE DISQUETTE AMSTRAD CPC (Tome 10)

Tout sur la programmation et la gestion des données avec le 6128 DDI-1 et le 664! Utile au débutant comme au programmeur en langage machine. Contient le listing du DOS commenté, un utilitaire qui ajoute les fichiers RELATIFS à l'AMDOS avec de nouvelles commandes BASIC, un MONITEUR disque et beaucoup d'autres programmes et astuces...



Réf. : ML 127
Prix : **149** FF



MONTAGES, EXTENSIONS ET PERIPHERIQUES AMSTRAD CPC (Tome 11)

Pour tous les amateurs d'électronique, ce livre montre ce que l'on peut réaliser avec un CPC. De nombreux schémas et exemples illustrent les thèmes et applications abordés comme les interfaces, programmateur d'EPROM... Un très beau livre de 450 pages.

Réf. : ML 131
Prix : **199** FF

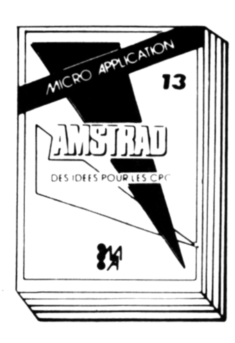


LE LIVRE DU CP/M AMSTRAD (Tome 12)

Ce livre vous permettra d'utiliser CP/M sur les CPC **464**, **664** et **6128** sans aucune difficulté. Vous y trouverez de nombreuses explications et les différents exemples vous assureront une maîtrise parfaite de ce très puissant système d'exploitation qu'est CP/M.

Réf. : ML 128
Prix : **149** FF

les livres Amstrad



DES IDÉES POUR LES CPC (Tome 13)

Vous n'avez pas d'idées pour utiliser votre CPC (464, 664, 6128)? Ce livre va vous en donner! Vous trouverez de très nombreux programmes BASIC couvrant des sujets très variés qui transformeront votre CPC en un bon petit génie. De plus les programmes vous permettront d'approfondir vos connaissances en programmation.

Réf. : ML 132
Prix : **129** FF



LES ROUTINES DE L'AMSTRAD CPC (Tome 14)

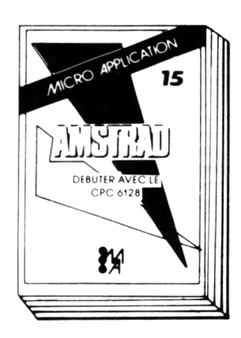
Pour bien connaitre et utiliser les routines utiles de l'AMSTRAD 6128, 664, 464. A la portée de tous. Nombreux programmes utilitaires, exemples, desassembleur, etc.

Réf. : ML 143
Prix : **149** FF

DÉBUTER AVEC LE CPC 6128 (tome 15)

Ce livre s'adresse à ceux qui débutent avec le CPC 6128. Tout leur est clairement expliqué aussi bien pour le matériel que pour le logiciel. Une fois leur machine bien en main, ils pourront s'attaquer au Basic et utiliser l'utilitaire de gestion d'adresses que contient le livre.

Réf. : ML 145
Prix : **99** F TTC

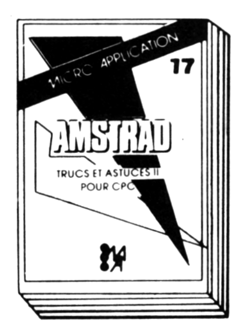


LA BIBLE DES CPC 664/6128 (tome 16)

Un régal pour tous ceux qui veulent tout connaitre sur les CPC 6128 et 664. Analyse du système d'exploitation, du processeur, le GATE ARRAY, le contrôleur video, le 8255, le chip sonore, les interfaces... Comprend un desassembleur, les points d'entrée des routines commentés de l'interpréteur et du système d'exploitation. Un super livre comme toutes les Bibles !

Réf. : ML 146 Prix : **199** F





TRUCS ET ASTUCES II POUR CPC

(tome 17)

Ce livre concerne tous les possesseurs de CPC (464, 664 et, bien sûr 6128 !). Vous y trouverez un générateur de menus, un générateur de masques, des aides à la programmation comme un DUMP, l'utilisation des routines systèmes et plein d'astuces de programmation. Pour tous ceux qui veulent tirer le maximum de leur CPC !

Réf. : ML 147
Prix : **129** F TTC

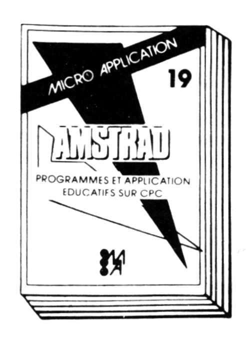


Réf. : ML 148
Prix : **149** FF
Disponible en Mai

LE LIVRE DE LA CAO (Tome 18)

Avec cet ouvrage vous saurez tout sur la Conception Assistée par Ordinateur et sur la programmation des GRAPHIQUES en 3 dimensions sur les CPC. Les points, lignes, rectangles, cercles, courbes, figures en 3D (comme les cubes, pyramides, cylindres, etc.), les rotations, les effets miroirs, les éclatements et explosions, et enfin pour conclure le clou : toutes les astuces pour créer son propre système de CAO. Nombreux programmes exemples et utilitaires.

les plus de Micro Application



PROGRAMMES et APPLICATIONS ÉDUCATIFS sur CPC. (Tome 19)

Ce livre est un recueil complet de programmes complets et d'applications prêts à fonctionner sur CPC. Chaque programme est très bien commenté et l'ouvrage couvre de nombreux sujets (mathématiques, chimie...). Ce livre est tout particulièrement destiné aux lycéens.
Réf : ML 150
Prix : **179** FF



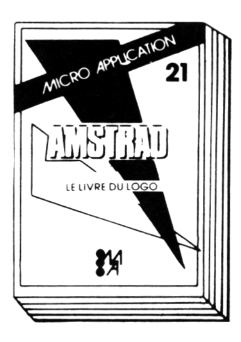
Disponible en Mai

SYSTÈMES DE TRANSMISSION SUR CPC. (Tome 20)

Encore une exclusivité Micro Application. Grâce à ce livre les communications et transmissions n'auront plus de secrets pour vous et vous pourrez profiter au maximum des possibilités offertes aujourd'hui dans ce domaine. Complet avec beaucoup d'applications pratiques, un ouvrage pratique et original.
Réf : ML 151
Prix : **199** F

LE LIVRE DU LOGO (Tome 21)

Le LOGO est un langage très intéressant dont les applications sont très nombreuses. Cet ouvrage permettra au lecteur de profiter au maximum du LOGO livré avec L'AMSTRAD. Principaux thèmes abordés : les graphismes, les procédures, les récursions, les routines de tri, un générateur de masque, structure des données, intelligence artificielle...
Ref. : ML 162
Prix : **149** FF
Disponible en Juin



INTELLIGENCE ARTIFICIELLE ET ROBOTIQUE SUR CPC (Tome 22)

Ce livre est une excellente introduction au monde de l'intelligence artificielle et à ses applications. Toutes les techniques et methodes décrites sont illustrees de programmes exemples. On apprendra ainsi quelle méthode un robot utilise pour trouver la sortie d'un labyrinthe ou comment un ordinateur peut acquérir des connaissances et ainsi aider a la résolution de problemes.



Ref. : ML 163
Prix : **149** FF
Disponible en Juin



BIEN DEBUTER AVEC LE PCW

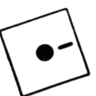
Le premier livre pour l'AMSTRAD PCW! Cet ouvrage vous permettra de réussir à coup sûr vos débuts sur le PCW. On découvre pas à pas le puissant taitement de texte LOCOSCRIPT, puis la programmation BASIC MALLARD et l'utilisation de CP/M. Indispensable pour bien profiter de son PCW.
Réf. : ML 164
Prix : **129** FF



Réf. : ML 165
Prix : **179** FF
Disponible en Juin

LE LIVRE DE L'AMSTRAD PCW

Vous possédez un PCW et vous voulez en tirer le maximum? Alors ce livre a été écrit pour vous! Grâce à lui vous utiliserez au mieux le LOCOSCRIPT et profiterez de toutes les possibilités offertes par le CP/M. Une formation intensive au BASIC MALLARD vous permettra d'écrire des routines d'édition, un générateur de masques de saisie, des routines de tri et une gestion de fichier.



DATAMAT

DATAMAT permet de tenir à jour et d'exploiter tous vos fichiers. De plus relié à CALCUMAT vous pourrez reprendre les données de vos fichiers pour établir des calculs et des graphes (par exemple : répartition géographique de vos clients, histogramme des ventes..). Relié à TEXTOMAT vous pourrez intégrer vos données pour réaliser des mailings, courrier personnalisé etc...

Spécifications techniques :

- Emploi extrèmement simple du à l'utilisation de menus.
- Traite tout type de données.
- Définition d'un masque de saisie personnalisé.
- 40 ou 80 caractères par ligne.
- Fonction de recopie d'écran sur imprimante.
- 50 champs par enregistrement.
- 512 caractères par enregistrement
- Jusqu'à 4000 enregistrements par fichier.
- Définition des couleurs écran-bordure- caractère .
- Utilisation des fichiers avec TEXTOMAT (mailing, relances...).
- Fonctionne avec un ou deux lecteurs de disquettes.
- Entièrement écrit en langage machine : extrèmement rapide.
- Adaptable à tout type d'imprimante.
- Jeu de caractères français accentué complet (ou anglais).
- Programme principal en mémoire : pas d'attente de chargement.
- Manuel d'utilisation complet en français.
- Impression d'étiquettes.
- DATAMAT fonctionne sur CPC 464, 664 et 6128.

REF : AM304
PRIX : 450 FF Version disquette



TEXTOMAT

Un Traitement de Texte puissant et simple qui tire partie de toutes les capacités des CPC.

TEXTOMAT vous permettra d'écrire, d'archiver et de modifier vos courriers , rapports , thèses , études... Vous pourrez intégrer dans vos documents des données extraites des fichiers DATAMAT et des calculs réalisés par CALCUMAT.

- Utilisation aisée à partir de menus.
- Jeu de caractères français complet accentué.
- Fonction de calcul en mode texte.
- Jusqu'à 16640 caractères.
- Possibilité de chainage de textes sur disquette.
- Fonctionne en mode 80 caractères avec accents.
- Travaille avec un ou deux lecteurs de disquettes.
- Choix des couleurs écran-caractères-bordure.
- Mode Insertion-Gomme....
- Tabulation.
- Numérotation des pages.
- Impression proportionnelle avec table d'espacements redéfinissable.
- Caractères de contrôle librement définissable (soulignage, double épaisseur..).
- Lettre type avec insertion automatique (adresses par exemple).
- Formatage des textes à l'écran.
- Adaptation à tout type d'imprimante.
- Manuel détaillé et didactique.
- Fonctionne sur 464, 664 et 6128.

SPECIFICATIONS TECHNIQUES :

3 Modes :
- Mode Texte
- Mode Commande : 13 ordres
- Mode Menu : 33 ordres

REF : AM305
PRIX: 450 FF Version disquette



CALCUMAT

CALCUMAT est un tableur graphique de qualité professionnelle. Il se compose principalement d'une grille de calcul, d'un calepin, d'une calculatrice, d'un presse papier et d'un module permettant la représentation graphique d'un ensemble de données.
CALCUMAT s'utilise très simplement à l'aide de menus déroulants et de fenêtres de travail.

PRINCIPALES FONCTIONS DE CALCUMAT

- Tri numérique ou alphanumérique d'un ensemble de cellule.

- Fonctions "couper, copier, coller" pour manipuler un ensemble de cellules par l'intermédiaire du presse papier.

- Calculs en mode automatique ou sur demande.

- Représentation graphique en barres, lignes, ou camembert, de quatre zones de données.

- Recopie d'écran graphique sur imprimante AMSTRAD DMP 1, DMP 2000, et compatibles Epson.

- Transfert de données de DATAMAT vers CALCUMAT pour effectuer des calculs sur les zones numériques d'un fichier.

- Transfert de données de CALCUMAT vers TEXTOMAT pour impressions de lettres circulaires.

- Calepin avec éditeur plein écran permettant l'impression de textes et de valeurs contenues dans la grille de calcul.

CARACTERISTIQUES DE CALCUMAT

- Grille de calcul de 256 colonnes sur 1024 lignes maximum.
- Capacité de 20 Ko de mémoire pour le stockage des données.
- Longueur maximum d'une formule de calcul de 100 caractères.
- Largeur des colonnes redéfinissable séparément.

REF : AM311
PRIX : 450 F Version disquette.



SPACE MOVING

SPACE MOVING est un utilitaire de création et d'animation de figures dans un espace à trois dimensions.

Il contient 3 programmes distincts:

"SPACE" est le programme de base qui permet la création et l'animation des figures. L'écran est divisé en deux zones, une fenêtre dans laquelle apparaissent les menus et les commandes, et une fenêtre graphique d'exécution.
"SPACE" possède entre autre des fonctions de translations, de rotations, de zoom et de copie d'écran.
Il permet de créer des fichiers " figures " et des fichiers " animation " qui peuvent être exécutés par lui-même ou à partir d'un programme BASIC.

"SPACE 3" est un programme intégrant 20 nouvelles instructions (RSX) au basic. Ces instructions permettent de créer et d'animer des figures en trois dimensions et d'utiliser les fichiers "figures" et " animation " crées par le programme "SPACE".

"DEMO" est le programme de démonstration graphique, en basic, qui utilise les différents fichiers " figure " et " animation " qui se trouvent sur la disquette.

SPECIFICATIONS TECHNIQUES

- Les figures créées peuvent comprendre jusqu'à 100 points et 150 ordres.
- Il est possible d'enregistrer jusqu'à 1024 mouvements consécutifs.
- Chaque fichier est automatiquement indexé par SPACE à la lecture ou à la sauvegarde.
- L'utilitaire de recopie d'écran graphique est entièrement reconfigurable suivant l'imprimante utilisée.

REF : AM210
PRIX: 295 FF Version cassette

REF : AM310
PRIX: 395 FF Version disquette

les logiciels Amstrad

SUPERPAINT

Superpaint est un logiciel utilitaire de création graphique sur ordinateur.

Superpaint fait partie de la nouvelle génération de logiciels qui utilisent la technique des icônes et des menus déroulants. La sélection des outils, des motifs, des fonctions ainsi que le dessin lui-même se font à l'aide d'un joystick ou d'une sourie.
Superpaint travaille sur des feuilles de format A4 (21 x 29,7) et ne visualise qu'une partie du dessin à l'écran de travail.

PRINCIPAUX OUTILS DE SUPERPAINT

Pinceau, crayon, gomme, règle, pot de peinture, bombe à laquer, trace cercle et élipses, trace rectangles, trace polygones, curseur de texte...
Toutes les formes peuvent être coloriées ou tramées à l'aide des motifs.

PRINCIPALES FONCTIONS DE SUPERPAINT

- ZOOM d'une partie du dessin pour modifications de pixels au crayon.
- Saisie de figures au lasso pour déplacements, copies, rotations, effets miroir, contours, inversions, remplissages...
- Fonctions "couper,copier,coller" pour sauvegarde ou intégration de parties de dessins.
- Grille pour déplacement des outils pas à pas.
- Editeur de texte intégré avec plusieurs polices de caractères.
- Ecriture de tous les jeux de caractères en souligné, italique, double hauteur...
- Création de pinceaux et de motifs
- Fonction "montrer page" pour visualiser la totalité du document.
- Impression avec représentation des couleurs sur imprimante graphique.

CARACTERISTIQUES DE SUPERPAINT.

- 4 couleurs utilisables parmis une palette de 27.
- Possibilité de mélange des couleurs à l'aide de motifs spéciaux
- 9 outils de travail et 24 motifs redéfinissables.
- Stockage de 5 dessins par face de disquette.
- Résolution graphique du dessin de 512 points horizontaux sur 408 points verticaux.
- Résolution graphique sur papier de 1024 points horizontaux sur 816 points verticaux avec 4 niveaux de gris.

REF : AM309
PRIX : 395 FF version disquette



D.A.M.S.

D.A.M.S est un logiciel intégrant un assembleur, un moniteur et un désassembleur symbolique pour développer et mettre au point facilement des programmes en langage machine sur les micro ordinateurs AMSTRAD. Les trois modules sont co-résidents en mémoire ce qui assure une grande souplesse d'utilisation. Vous pouvez notamment utiliser un éditeur plein écran, un assembleur immédiat, un désassembleur symbolique, une trace et beaucoup d'autres fonctions très puissantes. D.A.M.S est entièrement relogeable et est bien évidemment écrit en langage machine.

- L'éditeur est du type plein écran et sans numéro de ligne. Des commandes spéciales permettent la recherche ou la substitution de mots ou de phrases, l'effacement, la duplication et le déplacement de blocs de textes.

- L'assembleur Z80 est doté de pseudos instructions d'assemblage telles que IF, THEN, ELSE, et DEFB, DEFW etc...

Il permet l'assemblage de plusieurs blocs de texte source. La table des labels générée peut être ensuite utilisée par le moniteur, le désassembleur et le mode trace.

- Le moniteur comprend plus de 15 commandes pour utiliser trace,dump, affichage et modification des registres, exécution d'un programme...

- Le désassembleur peut créer du texte source à partir d'un programme en langage machine avec génération automatique de labels et DEFB.. Le source peut être modifié à partir de l'éditeur et réassemblé.

- La trace comprend un mode rapide pour mettre au point des routines importantes. Chaque instruction est analysée avant son exécution pour éviter tout blocage système.

D.A.M.S est entièrement autonome (il possède sa propre pile et ses routines systèmes).

REF : AM208
PRIX : 295 FF Version cassette
REF : AM308
PRIX : 395 FF Version Disquette

AUTOFORMATION A L'ASSEMBLEUR

LE LANGAGE MACHINE A LA PORTEE DE TOUS
Contient un livre et un logiciel.

Le livre :

Cet ouvrage introduit le débutant à la programmation du Z80 en utilisant la méthode du Dr WATSON qui selon les critiques vaut son pesant d'or !
Aucune connaissance préalable n'est requise et le but du livre est d'assurer au novice un succès total. A la fin du livre les instructions du Z80 sont expliquées en détail. De nombreux exemples illustrent les différentes étapes au cours alors que les exercices (les solutions sont fournies) testent la compréhension. D'autres chapitres montrent comment de nouvelles commandes peuvent être ajoutées au BASIC, notamment une routine de traçage de CERCLE.

Le logiciel :

Un assembleur Z80 complet est livré sur cassette et comprend :

- Etiquettes Symboliques.
- Directives d'Assemblage.
- Chargement/Sauvegarde.
- Copie d'Ecran.
- INSERT/DELET.

L'assembleur permet d'écrire des programmes facilement en langage d'assemblage puis les transforme en code machine (langage machine).

Pour vous aider à comprendre les notations mathématiques utilisées, une démonstration de l'utilisation des nombres binaires et hexadécimaux est fournie.
Un programme utilisant les commandes graphiques additionnelles décrites dans le livre est également fourni.

Réf : ML226 Prix : 195 F TTC
version cassette.
REF : ML326 Prix : 295 F TTC
version disquette.

AVEZ-VOUS LU MICRO INFO ?


MICRO INFO
123
X82
ATARI
COMMODORE
AMSTRAD
80 pages de trucs et astuces,
programmes, dossiers
pour Amstrad CPC,
Commodore
Atari ST ...
20 f

MICRO APPLICATION vous présente MICRO INFO, nouveau journal avec des dossiers, des bidouilles, des trucs et astuces, des nouveautés, des programmes et plein de rubriques sympas! (88 pages)
Chaque numéro traite principalement de 3 matériels:

AMSTRAD - COMMODORE - ATARI

carte d'abonnement

Je désire m'abonner à MICRO INFO

☐ Le numéro 1 : 15 F + 5 F pour frais d'envoi
☐ Le numéro 2 : 20 F + 5 F pour frais d'envoi
☐ Les numéros 1 et 2 : 35 F + 5 F pour frais d'envoi
☐ Je choisis de m'abonner pour 4 numéros au prix de 70F

Je règle par ☐ chèque
☐ mandat
☐ CCP

Nom : ________________ Prénom : ________________
Adresse : ________________________________
Code postal : ________________ date et signature :

Veuillez nous retourner cette carte sous pli ainsi que votre règlement à l'adresse suivante :

MICRO APPLICATION
13 rue Sainte Cécile 75009 PARIS

Le LOGO est un langage aux multiples possibilités et ouvre de nombreux horizons à ses utilisateurs. Certains seront séduits par sa simplicité d'utilisation, d'autres y trouveront des solutions dans le domaine de l'intelligence artificielle et bien sûr tous apprécieront d'utiliser pleinement les capacités graphiques de l'Amstrad grâce à LOGO. Ce livre comporte de nombreux exemples, illustrations et exercices nécessaires à une bonne pratique du LOGO en intégrant toutes les particularités de l'AMSTRAD.

Quelques-uns des thèmes abordés :

- Le calcul sous LOGO.
- Le hasard et les répétitions.
- Le maniement des procédures.
- La programmation graphique et les récursions.
- Toutes les instructions du système LOGO.
- Les mots et les listes.
- Les instructions d'entrée et de sortie.
- Les structures de données en LOGO.
- Les programmes en tant que listes.
- Le traitement des erreurs.
- Les listes de propriétés.
- La gestion de fichier sous LOGO.
- Programmation des couleurs et du son.

Ce livre fonctionne pour le CPC 464 et 6128 ainsi que le PCW 8256 et 8512.

Prix : 149 FF
Réf. : ML 162

ISBN : 2-86899-072-X

AMSTRAD LE LIVRE DU LOGO PCW et CPC





Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/